

PA01-308

reference 4

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平10-340276

(43) 公開日 平成10年(1998)12月22日

(51) Int.Cl. ⁴	識別記号	F I	
G 0 6 F 17/30		G 0 6 F 15/403	3 4 0 D
17/18		15/38	Z
		15/40	3 1 0 C
			3 8 0 D

審査請求 有 請求項の数18 O L (全 28 頁)

(21) 出願番号 特願平9-339185

(22) 出願日 平成9年(1997)12月9日

(31) 優先権主張番号 特願平9-90384

(32) 優先日 平9(1997)4月9日

(33) 優先権主張国 日本 (J P)

(71) 出願人 592073101

日本アイ・ピー・エム株式会社
東京都港区六本木3丁目2番12号

(72) 発明者 松澤 裕史

神奈川県大和市下鶴間1823番地14 日本ア
イ・ピー・エム株式会社東京基礎研究所内

(72) 発明者 福田 剛志

神奈川県大和市下鶴間1823番地14 日本ア
イ・ピー・エム株式会社東京基礎研究所内

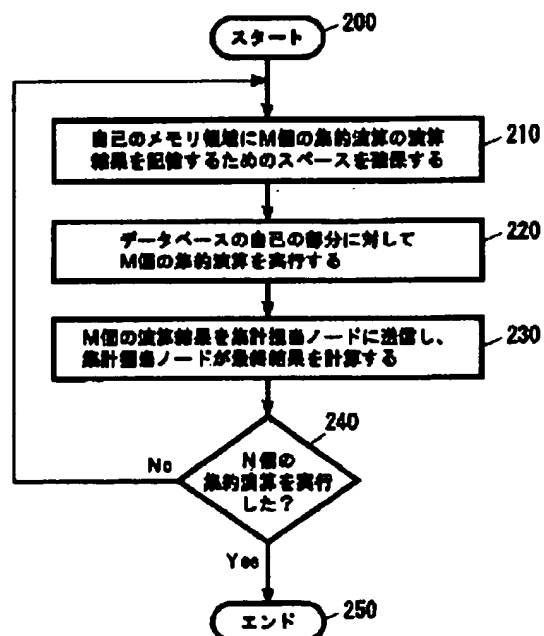
(74) 代理人 弁理士 坂口 博 (外1名)

(54) 【発明の名称】 集約演算実行方法及びコンピュータ・システム

(57) 【要約】

【課題】 複数の集約演算を並列に高速実行する方法を提供すること

【解決手段】 ネットワークにより接続された複数のプロセッサが各々自己のメモリ領域及び1又は複数のグループに分けることができるデータを含むデータベースの自己の部分を使用できるよう構成されたコンピュータ・システムにおいて、(a) 自己のメモリ領域に、N個の集約演算のうちM個(N以下の整数)分の演算結果を記憶するスペースを確保するステップと、(b) M個の集約演算をまとめて、データベースの自己の部分に対して実行するステップと、(c) M個の集約演算の各プロセッサにおける演算結果を、集計すべきプロセッサに送信し、集計すべきプロセッサが最終結果を計算するステップと、(d) N個の集約演算が終了するまで、ステップ(a)乃至(c)を繰り返すステップとを含む。



【特許請求の範囲】

【請求項1】 ネットワークにより接続された複数のプロセッサが各々自己のメモリ領域及び1又は複数のグループに分けることができるデータを含むデータベースの自己の部分を使用できるよう構成されたコンピュータ・システムにおいて、前記グループ毎に集約を行う演算を含む、N個の集約演算を前記データベースに対し実行する方法であって、(a) 各前記プロセッサが、前記自己のメモリ領域に、前記N個の集約演算のうちM個(N以下の整数)分の演算結果を記憶するスペースを確保するステップと、(b) 各前記プロセッサが、M個の集約演算をまとめて、前記データベースの自己の部分に対して実行するステップと、(c) 各前記プロセッサが、前記M個の集約演算の各プロセッサにおける演算結果を、集計すべきプロセッサに送信し、前記集計すべきプロセッサが最終結果を計算するステップと、(d) 前記N個の集約演算の実行が終了するまで、前記ステップ(a)乃至(c)を繰り返すステップとを含む集約演算実行方法。

【請求項2】 前記ステップ(b)が、(b1) 各前記プロセッサが、前記自己のメモリ領域のワークスペースに、前記データベースの自己の部分である部分データベースの一部を読み出すステップと、(b2) 各前記プロセッサが、前記自己のメモリ領域に格納された本ステップ以前の演算結果と読み出された前記部分データベースの一部とについて、前記M個の集約演算を実行するステップと、(b3) 前記部分データベースのすべてに対して前記M個の集約演算が実行されるまで、前記ステップ(b1)及び(b2)を繰り返すステップとを含む請求項1記載の集約演算実行方法。

【請求項3】 前記Mは集約演算の演算結果のために前記自己のメモリ領域に確保できるスペースから決定されることを特徴とする請求項1記載の集約演算実行方法。

【請求項4】 ネットワークにより接続された複数のプロセッサが各々自己のメモリ領域及び1又は複数のグループに分けることができるデータを含むデータベースの自己の部分を使用できるよう構成されたコンピュータ・システムにおいて、前記グループ毎に集約を行う演算を含む、P個の集約演算を前記データベースに対し実行する方法であって、(a) 各前記プロセッサが、前記P個の集約演算のうち自らが実行するQ個の集約演算の演算結果を記憶するスペースを、前記自己のメモリ領域に確保するステップと、(b) 各前記プロセッサが、前記自己のメモリ領域に前記データベースの自己の部分である部分データベースの一部を読み出し、読み出された前記部分データベースの一部を前記ネットワークを介してブロードキャストすることを繰り返して、各前記プロセッサが全ての前記データベースのデータに対し前記自らが実行するQ個の集約演算を実行するステップと、(c) 前記P個の集約演算が実行されるまで、前記ステップ

(a) 及び(b)を繰り返すステップとを含む集約演算

実行方法。

【請求項5】 前記ステップ(a)が、(a1) 1つの集約演算の演算結果を格納できるようなスペースが1つのプロセッサの前記自己のメモリ領域に存在するか検査するステップと、(a2) 前記スペースが存在する場合には、前記スペースを前記1つの集約演算の演算結果のために確保するステップと、(a3) 前記スペースが存在しない場合には、他のプロセッサの前記メモリ領域に前記1つの集約演算の演算結果を格納できるようなスペースが存在するか検査するステップと、(a4) 前記他のプロセッサの前記メモリ領域にスペースが存在する場合には、当該他のプロセッサのメモリ領域のスペースを前記1つの集約演算の演算結果のために確保するステップとを含み、

前記他のプロセッサのメモリ領域にスペースが存在しない場合には、後の繰り返し処理にて前記1つの集約演算を実行することとを特徴とする請求項4記載の集約演算実行方法。

【請求項6】 前記Qは集約演算の演算結果のために前記自己のメモリ領域に確保できるスペースから決定されることを特徴とする請求項4記載の集約演算実行方法。

【請求項7】 前記ステップ(b)が、(b1) 各前記プロセッサが、前記自己のメモリ領域のワークスペースに、前記部分データベースの一部を読み出すステップと、

(b2) 読み出された前記部分データベースの一部を前記ネットワークを介してブロードキャストするステップと、(b3) 前記自己のメモリ領域に格納された本ステップ以前の演算結果と前記読み出された部分データベースの一部と他のプロセッサから送られてきたデータとについて、前記自らが実行するQ個の集約演算を実行するステップと、(b4) 各前記プロセッサが全ての前記データベースの内容に対し前記自らが実行するQ個の集約演算を実行するまで前記ステップ(b1)乃至(b3)を繰り返すステップとを含む請求項4記載の集約演算実行方法。

【請求項8】 ネットワークにより接続された複数のプロセッサが各々自己のメモリ領域及び1又は複数のグループに分けることができるデータを含むデータベースの自己の部分を使用できるよう構成されたコンピュータ・システムにおいて、前記グループ毎に集約を行う演算を含む、S個の集約演算を前記データベースに対し実行する方法であって、(a) 前記S個の集約演算のうち実行するT個の集約演算を決定するステップと、(b) 前記T個の集約演算を実行する際に集約される、各集約演算の各グループを取り扱うプロセッサを決定するステップと、(c) 各前記プロセッサが、前記データベースの自己の部分である部分データベースの一部を前記自己のメモリ領域に読み出し、読み出されたデータのうち他のプロセッサが集約すべき集約演算のグループに関するデータを当該集約演算のIDと共に前記他のプロセッサに前記ネットワークを介して送信し、自己が集約すべき集約

演算のグループに関するデータについて前記T個の集約演算を実行するステップと、(d)各前記プロセッサが集約すべき集約演算のグループに関する全てのデータに対して前記T個の集約演算を実行するまで、前記ステップ(c)を実行するステップと、(e)前記S個の集約演算を実行するまで、前記ステップ(a)乃至(d)を繰り返すステップとを含む集約演算実行方法。

【請求項9】前記ステップ(c)が、(c1)各前記プロセッサが、前記データベースの自己の部分である部分データベースの一部を前記自己のメモリ領域のワークスペースに読み出すステップと、(c2)各前記プロセッサが、読み出されたデータの各部分を必要とするプロセッサを求め、当該部分を、関連する集約演算のIDと共に前記必要とするプロセッサに前記ネットワークを介して送信するステップと、(c3)各前記プロセッサが、前記読み出されたデータのうち自己が集約すべき集約演算のグループに関するデータ及び前記他のプロセッサからのデータ及び前記自己のメモリ領域に格納された本ステップ以前の演算結果に対し、前記T個の集約演算を実行するステップと、

を含む請求項8記載の集約演算実行方法。

【請求項10】ネットワークにより接続された複数のプロセッサが各々自己のメモリ領域及び1又は複数のグループに分けることができるデータを含むデータベースの自己の部分を使用できるよう構成されたコンピュータ・システムにおいて、前記グループ毎に集約を行う演算を含む、N個の集約演算を前記データベースに対し実行する方法であって、

前記プロセッサの数と前記データベースの大きさと前記ネットワークの通信速度とを含む前記コンピュータ・システムのパラメータ及び実行する集約演算の数と各集約演算の結果を格納するメモリの量とを含む集約演算の性質に関するパラメータから、(a)各前記プロセッサが、前記自己のメモリ領域に、前記N個の集約演算のうちM個(N以下の整数)分の演算結果を記憶するスペースを確保するステップと、(b)各前記プロセッサが、M個の集約演算をまとめて、前記データベースの自己の部分に対して実行するステップと、(c)各前記プロセッサが、前記M個の集約演算の各プロセッサにおける演算結果を、集計すべきプロセッサに送信し、前記集計すべきプロセッサが最終結果を計算するステップと、

(d)前記N個の集約演算の実行が終了するまで、前記ステップ(a)乃至(c)を繰り返すステップとを含む第1集約演算実行方法と、(e)各前記プロセッサが、前記N個の集約演算のうち自らが実行するQ個の集約演算の演算結果を記憶するスペースを、前記自己のメモリ領域に確保するステップと、(f)各前記プロセッサが、前記自己のメモリ領域に前記データベースの自己の部分である部分データベースの一部を読み出し、読み出された前記部分データベースの一部を前記ネットワーク

を介してブロードキャストすることを繰り返して、各前記プロセッサが全ての前記データベースのデータに対し前記自らが実行するQ個の集約演算を実行するステップと、(g)前記N個の集約演算が実行されるまで、前記ステップ(e)及び(f)を繰り返すステップとを含む第2集約演算実行方法と、(h)前記N個の集約演算のうち実行するT個の集約演算を決定するステップと、

(i)前記T個の集約演算を実行する際に集約される、各集約演算の各グループを取り扱うプロセッサを決定するステップと、(j)各前記プロセッサが、前記データベースの自己の部分である部分データベースの一部を前記自己のメモリ領域に読み出し、読み出されたデータのうちの他のプロセッサが集約すべき集約演算のグループに関するデータを当該集約演算のIDと共に前記他のプロセッサに前記ネットワークを介して送信し、自己が集約すべき集約演算のグループに関するデータについて前記T個の集約演算を実行するステップと、(k)各前記プロセッサが集約すべき集約演算のグループに関する全てのデータに前記T個の集約演算を実行するまで、前記ステップ(j)を実行するステップと、(l)前記N個の集約演算を実行するまで、前記ステップ(h)乃至

(k)を繰り返すステップとを含む第3集約演算実行方法とを含む複数の集約演算実行方法のうちいずれの方法が最も高速に実行できるかを決定するステップと、決定された方法にて、前記N個の集約演算を実行するステップとを含む集約演算実行方法。

【請求項11】ネットワークにより接続された複数のプロセッサが各々自己のメモリ領域及び1又は複数のグループに分けることができるデータを含むデータベースの自己の部分を使用できるよう構成されたコンピュータ・システムであって、

前記グループ毎に集約を行う演算を含む、N個の集約演算を前記データベースに対し実行するために、前記複数のプロセッサの各々は、(a)前記自己のメモリ領域に、前記N個の集約演算のうちM個(N以下の整数)分の演算結果を記憶するスペースを確保するメモリ処理装置と、(b)M個の集約演算をまとめて、前記データベースの自己の部分に対して実行するデータベース処理装置と、(c)前記M個の集約演算の演算結果を、集計すべきプロセッサに送信する送信機と、

を有し、前記N個の集約演算の実行が終了するまで処理を繰り返し、前記集計すべきプロセッサは自己が集計すべき集約演算の演算結果を他のプロセッサから受信し、集計することを特徴とするコンピュータ・システム。

【請求項12】ネットワークにより接続された複数のプロセッサが各々自己のメモリ領域及び1又は複数のグループに分けることができるデータを含むデータベースの自己の部分を使用できるよう構成されたコンピュータ・システムであって、

前記グループ毎に集約を行う演算を含む、P個の集約演

算を前記データベースに対し実行するために、前記複数のプロセッサの各々は、(a) 前記P個の集約演算のうち自らが実行するQ個の集約演算の演算結果を記憶するスペースを、前記自己のメモリ領域に確保するメモリ処理装置と、(b) 前記自己のメモリ領域のワークスペースに前記データベースの自己の部分である部分データベースの一部を読み出し、読み出された前記部分データベースの一部を前記ネットワークを介してブロードキャストすることを繰り返して、各前記プロセッサが全ての前記データベースのデータに対し前記自らが実行するQ個の集約演算を実行するデータベース処理装置と、を有し、前記P個の集約演算が実行されるまで処理を繰り返すことを特徴とするコンピュータ・システム。

【請求項13】 ネットワークにより接続された複数のプロセッサが各々自己のメモリ領域及び1又は複数のグループに分けることができるデータを含むデータベースの自己の部分を使用できるよう構成されたコンピュータ・システムであって、

前記グループ毎に集約を行う演算を含む、S個の集約演算を前記データベースに対し実行するために、

前記S個の集約演算のうち実行するT個の集約演算を決定し、当該T個の集約演算を実行する際に集約される、各集約演算の各グループを取り扱うプロセッサを決定するコントローラを含み、

各前記プロセッサは、

前記データベースの自己の部分である部分データベースの一部を読み出し、読み出されたデータのうちの他のプロセッサが集約すべき集約演算のグループに関するデータを当該集約演算のIDと共に前記他のプロセッサに前記ネットワークを介して送信し、自己が集約すべき集約演算のグループに関するデータについて前記T個の集約演算を実行するデータベース処理装置を有し、

各前記プロセッサが集約すべき集約演算のグループに関する全てのデータに前記T個の集約演算を実行するまで前記データベース処理装置を動作させ、

前記S個の集約演算を実行するまで、前記コントローラ及び各前記プロセッサを動作させることを特徴とするコンピュータ・システム。

【請求項14】 ネットワークにより接続された複数のプロセッサが各々自己のメモリ領域及び1又は複数のグループに分けることができるデータを含むデータベースの自己の部分を使用できるよう構成されたコンピュータ・システムであって、

前記グループ毎に集約を行う演算を含む、N個の集約演算を前記データベースに対し実行するために、

前記プロセッサの数と前記データベースの大きさと前記ネットワークの通信速度とを含む前記コンピュータ・システムのパラメータ及び実行する集約演算の数と各集約演算の結果を格納するメモリの量とを含む集約演算の性質に関するパラメータから、(a) 各前記プロセッサ

が、前記自己のメモリ領域に、前記N個の集約演算のうちM個(N以下の整数)分の演算結果を記憶するスペースを確保するステップと、(b) 各前記プロセッサが、M個の集約演算をまとめて、前記データベースの自己の部分に対して実行するステップと、(c) 各前記プロセッサが、前記M個の集約演算の各プロセッサにおける演算結果を、集計すべきプロセッサに送信し、前記集計すべきプロセッサが最終結果を計算するステップと、

(d) 前記N個の集約演算の実行が終了するまで、前記ステップ(a)乃至(c)を繰り返すステップとを含む第1集約演算実行方法と、(e) 各前記プロセッサが、前記N個の集約演算のうち自らが実行するQ個の集約演算の演算結果を記憶するスペースを、前記自己のメモリ領域に確保するステップと、(f) 各前記プロセッサが、前記自己のメモリ領域に前記データベースの自己の部分である部分データベースの一部を読み出し、読み出された前記部分データベースの一部を前記ネットワークを介してブロードキャストすることを繰り返して、各前記プロセッサが全ての前記データベースのデータに対し前記自らが実行するQ個の集約演算を実行するステップと、(g) 前記N個の集約演算が実行されるまで、前記ステップ(e)及び(f)を繰り返すステップとを含む第2集約演算実行方法と、(h) 前記N個の集約演算のうち実行するT個の集約演算を決定するステップと、

(i) 前記T個の集約演算を実行する際に集約される、各集約演算の各グループを取り扱うプロセッサを決定するステップと、(j) 各前記プロセッサが、前記データベースの自己の部分である部分データベースの一部を前記自己のメモリ領域に読み出し、読み出されたデータのうちの他のプロセッサが集約すべき集約演算のグループに関するデータを当該集約演算のIDと共に前記他のプロセッサに前記ネットワークを介して送信し、自己が集約すべき集約演算のグループに関するデータについて前記T個の集約演算を実行するステップと、(k) 各前記プロセッサが集約すべき集約演算のグループに関する全てのデータに前記T個の集約演算を実行するまで、前記ステップ(j)を実行するステップと、(l) 前記N個の集約演算を実行するまで、前記ステップ(h)乃至

(k)を繰り返すステップとを含む第3集約演算実行方法とを含む複数の集約演算実行方法のうちいずれの方法が最も高速に実行できるかを決定するセレクトと、決定された方法にて、前記N個の集約演算を実行するよう命ずる手段とを有するコンピュータ・システム。

【請求項15】 ネットワークにより接続された複数のプロセッサが各々自己のメモリ領域及び1又は複数のグループに分けることができるデータを含むデータベースの自己の部分を使用できるよう構成されたコンピュータ・システムに、前記グループ毎に集約を行う演算を含む、N個の集約演算を前記データベースに対し実行させるプログラムを格納した記憶媒体であって、

前記プログラムは、(a) 各前記プロセッサが、前記自己のメモリ領域に、前記N個の集約演算のうちM個(N以下の整数)分の演算結果を記憶するスペースを確保するステップと、(b) 各前記プロセッサが、M個の集約演算を同時に、前記データベースの自己の部分に対して実行するステップと、(c) 各前記プロセッサが、前記M個の集約演算の各プロセッサにおける演算結果を、集計すべきプロセッサに送信し、前記集計すべきプロセッサが最終結果を計算するステップと、(d) 前記N個の集約演算が終了するまで、前記ステップ(a)乃至(c)を繰り返すステップとを前記コンピュータ・システムに実行させる、記憶媒体。

【請求項16】ネットワークにより接続された複数のプロセッサが各々自己のメモリ領域及び1又は複数のグループに分けることができるデータを含むデータベースの自己の部分を使用できるよう構成されたコンピュータ・システムに、前記グループ毎に集約を行う演算を含む、P個の集約演算を前記データベースに対し実行させるプログラムを記憶した記憶媒体であって、

前記プログラムは、(a) 各前記プロセッサが、前記P個の集約演算のうち自らが実行するQ個の集約演算の演算結果を記憶するスペースを、前記自己のメモリ領域に確保するステップと、(b) 各前記プロセッサが、前記自己のメモリ領域のワークスペースに前記データベースの自己の部分である部分データベースの一部を読み出し、読み出された前記部分データベースの一部を前記ネットワークを介してブロードキャストすることを繰り返して、各前記プロセッサが全ての前記データベースのデータに対し前記自らが実行するQ個の集約演算を実行するステップと、(c) 前記P個の集約演算が実行されるまで、前記ステップ(a)及び(b)を繰り返すステップとを前記コンピュータ・システムに実行させる、記憶媒体。

【請求項17】ネットワークにより接続された複数のプロセッサが各々自己のメモリ領域及び1又は複数のグループに分けることができるデータを含むデータベースの自己の部分を使用できるよう構成されたコンピュータ・システムに、前記グループ毎に集約を行う演算を含む、S個の集約演算を前記データベースに対し実行させるプログラムを格納した記憶媒体であって、

前記プログラムは、(a) 前記S個の集約演算のうち実行するT個の集約演算を決定するステップと、(b) 前記T個の集約演算を実行する際に集約される、各集約演算の各グループを取り扱うプロセッサを決定するステップと、(c) 各前記プロセッサが、前記データベースの自己の部分である部分データベースの一部を前記自己のメモリ領域に読み出し、読み出されたデータのうちの他のプロセッサが集約すべき集約演算のグループに関するデータを当該集約演算のIDと共に前記他のプロセッサに前記ネットワークを介して送信し、自己が集約すべき集

約演算のグループに関するデータについて前記T個の集約演算を実行するステップと、(d) 各前記プロセッサが集約すべき集約演算のグループに関する全てのデータに前記T個の集約演算を実行するまで、前記ステップ

(c)を実行するステップと、(e) 前記S個の集約演算を実行するまで、前記ステップ(a)乃至(d)を繰り返すステップとを前記コンピュータ・システムに実行させる、記憶媒体。

【請求項18】ネットワークにより接続された複数のプロセッサが各々自己のメモリ領域及び1又は複数のグループに分けることができるデータを含むデータベースの自己の部分を使用できるよう構成されたコンピュータ・システムに、前記グループ毎に集約を行う演算を含む、N個の集約演算を前記データベースに対し実行させるプログラムを格納した記憶媒体であって、

前記プログラムは、

前記プロセッサの数と前記データベースの大きさと前記ネットワークの通信速度とを含む前記コンピュータ・システムのパラメータ及び実行する集約演算の数と各集約演算の結果を格納するメモリの量とを含む集約演算の性質に関するパラメータから、(a) 各前記プロセッサが、前記自己のメモリ領域に、前記N個の集約演算のうちM個(N以下の整数)分の演算結果を記憶するスペースを確保するステップと、(b) 各前記プロセッサが、M個の集約演算をまとめて、前記データベースの自己の部分に対して実行するステップと、(c) 各前記プロセッサが、前記M個の集約演算の各プロセッサにおける演算結果を、集計すべきプロセッサに送信し、前記集計すべきプロセッサが最終結果を計算するステップと、

(d) 前記N個の集約演算の実行が終了するまで、前記ステップ(a)乃至(c)を繰り返すステップとを含む第1集約演算実行方法と、(e) 各前記プロセッサが、前記N個の集約演算のうち自らが実行するQ個の集約演算の演算結果を記憶するスペースを、前記自己のメモリ領域に確保するステップと、(f) 各前記プロセッサが、前記自己のメモリ領域に前記データベースの自己の部分である部分データベースの一部を読み出し、読み出された前記部分データベースの一部を前記ネットワークを介してブロードキャストすることを繰り返して、各前記プロセッサが全ての前記データベースのデータに対し前記自らが実行するQ個の集約演算を実行するステップと、(g) 前記N個の集約演算が実行されるまで、前記ステップ(e)及び(f)を繰り返すステップとを含む第2集約演算実行方法と、(h) 前記N個の集約演算のうち実行するT個の集約演算を決定するステップと、

(i) 前記T個の集約演算を実行する際に集約される、各集約演算の各グループを取り扱うプロセッサを決定するステップと、(j) 各前記プロセッサが、前記データベースの自己の部分である部分データベースの一部を前記自己のメモリ領域に読み出し、読み出されたデータの

うち他のプロセッサが集約すべき集約演算のグループに関するデータを当該集約演算のIDと共に前記他のプロセッサに前記ネットワークを介して送信し、自己が集約すべき集約演算のグループに関するデータについて前記T個の集約演算を実行するステップと、(k)各前記プロセッサが集約すべき集約演算のグループに関する全てのデータに前記T個の集約演算を実行するまで、前記ステップ(j)を実行するステップと、(l)前記N個の集約演算を実行するまで、前記ステップ(h)乃至(k)を繰り返すステップとを含む第3集約演算実行方法とを含む複数の集約演算実行方法のうちいずれの方法が最も高速に実行できるかを決定するステップと、決定された方法にて、前記N個の集約演算を実行するステップとを含む記憶媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、データ・マイニング(Data Mining)のようなデータベースの処理等に必要となる集約演算(アグリゲーション:aggregation)の並列計算機上における処理方法に関する。なお、集約演算にはデータベース内のグループ毎に集約を行う演算を含む。

【0002】

【従来の技術】データ・マイニングのような分野にてよく用いられる集約演算とは、関係データベースへの問い合わせ言語SQLのgroup-byのように、あるリレーションに対し、特定の属性が同一の値又同一の値の組若しくは所定の範囲に属する値を有するようなレコードを1つのグループとして、グループ毎に他の属性の合計、最大、最小、平均等を計算する処理である。このような集約演算は、最近、OLAP(On Line Analytical Processing)【詳しくは、E.F.Codd,S.B.Codd, and C.T.Sally, "Beyond decision support," Computerworld, 27(30), July 1993参照のこと】やデータ・マイニング等の決定支援システムにおいて不可欠な操作となっている。

【0003】以下集約演算の例を示す。表1は、データベースに記憶されている関係を表している。プロダクト番号、カスタマー番号、そして売上げについての列があり、各行は売上げが生ずるごとに記録されるようになっている。このような関係について、プロダクト番号ごとに売上げを合計するという演算を考える。この演算は、プロダクト番号ごとにグループが構成され、その各グループごとに合計を計算するという処理であり、上記集約演算の1つの例である。この集約演算を実施すると、表2のような結果を得ることができる。

【表1】

product#	customer#	sold
G 1	C 1	3
G 1	C 2	10
G 2	C 2	5
G 2	C 3	10
G 3	C 4	2

【表2】

product#	sold
G 1	8
G 2	25
G 3	2

【0004】以前より、1つの集約演算を並列処理するためのアルゴリズムについては研究されてきた。以下、それらのアルゴリズムについて説明する。但し、各アルゴリズムにおいては、データベース中の全関係は各プロセッサに均等分割されていることを前提としている。

【0005】1. 2P アルゴリズム

二段階にて実施されるため2Pという。

(1)第1フェーズとして、各プロセッサ(ノードともいう)は、そのプロセッサ用のディスク装置(データベースを記憶している)に対して、集約演算を実施する。

(2)第2フェーズとして、各プロセッサの結果は、集計用のプロセッサに集められ、最終結果を計算する。

【0006】このような方法は、特開平5-2610号公報にも記載されている。この公報では、ただか1つのグループについての集約演算しか取り扱っておらず、先に述べたようなデータ・マイニングのように複数のグループについて集約演算を実施することに何等の考慮もなされていない。

【0007】2. Rep アルゴリズム

リパーティション(repartition)アルゴリズムは以下のようなアルゴリズムにて実施される(図1)。

(1)最初に、各ノードが集約演算を実行すべきグループを決定する(ステップ110)。表1の例では、ノード1がプロダクト番号G1を担当し、ノード2がプロダクト番号G2を担当する、といったような割当てを決定する。

(2)次に、各ノードで、そのノード用のディスク装置(データベースを記憶した物)から、データの一部を読み出し、他のノードが集約すべきグループのデータならば、担当ノードに送信する(ステップ120)。例えば表1の例で、この表1がノード1用のディスク装置内に存在する場合、ノード1が表1の第2行目を読み出した時、ノード1は第2行目のデータをノード2に送信する。

(3)そして、各ノードで、他のノードから送られてきたデータを含む、自己が集約演算をすべきグループに関するデータに対して集約を実行する(ステップ130)。

【0008】このような2つのアルゴリズムのうちどちらを用いると高速に処理できるかは、条件によって異なる。各アルゴリズムを用いて、1つの集約演算を処理した場合の所要時間の見積りを図2に示す。図2は、16ノードのIBM SP2（インタナショナル・ビジネス・マシーニズ・コーポレーションの商標）を用いた場合の見積りで、グループ数と応答時間の関係を示している。このシステムの場合、グループ数が 2×10^5 より少ない場合には、2Pアルゴリズムの方が高速であるが、グループ数が多い場合には、Repアルゴリズムの方が高速である。これまで、1つの集約演算の並列処理には、2PアルゴリズムとRepアルゴリズムを動的に切り換えるようなアルゴリズムが提案されている（例えば、Ambuj Shatdal and Jeffrey F. Naughton, "Adaptive parallel aggregation algorithms," In Proceedings of the ACM SIGMOD Conference on Management of Data, pages 104-114, May 1995. を参照のこと）。

【0009】また、上記以外の方法としては、データベース中の全レコードをブロードキャストする方法がある（Dine Bitton, Haran Boral, David J DeWitt, and W. Kevin Wilkinson, "Parallel algorithms for the execution of relational database operations," ACM Trans. on Database Systems, 8(3):324-353, Sep. 1983. を参照のこと）。しかし、このようなアルゴリズム（以下、BCアルゴリズムと呼ぶ。）は、プロセッサ間を接続するネットワークが低速であった頃には非実用的な方法であった。

【0010】このBCアルゴリズムをまとめておく。

3. BC アルゴリズム

(1) 全グループの中で、各ノードが集約演算を実施すべきグループを決定する。Repアルゴリズムと同様である。

(2) 各ノードは、自己のディスク装置内の全データを他の全てのノードに、ブロードキャストする。

(3) 各ノードは、ブロードキャストされたデータ（自己のディスク装置内のデータを含む）に対して、自己が集約すべきグループについての集約演算を実施する。

【0011】これまでは、1つの集約演算を並列に実行するアルゴリズムを説明したが、1つのプロセッサで複数の集約演算を実行する場合には、演算の順番を調整したり、演算間の関連性を用いて全体の処理を高速化する方法（例えば、Sameet Agrawal, Rakesh Agrawal, Prasad M. Deshpande, Ashish Gupta, Jeffrey F. Naughton, Raghu Ramakrishnan, and Sunita Sarawagi, "On the computation of multidimensional aggregates," In Proceedings of the 22nd VLDB Conference, Sep. 1996. を参照のこと）等が提案されている。しかし、並列計算機を用いて複数の演算を同時に処理する方法については、提案されていない。

【0012】

【発明が解決しようとする課題】複数の集約演算を並列に実行する処理は、上記の方法を複数回繰り返せば高速に実行できる、というわけではない。以上のように本発明は、複数の集約演算を並列に高速実行する方法を提供することを目的とする。

【0013】現在注目を集めている決定支援システムのコアとなる技術であるOLAPやデータマイニングでは、複数の集約演算を行う必要がある。例えば、多次元属性を持つデータの分析に対して、複数の集約演算を実行するData Cube（例えば、Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh, "Data cube: A relational aggregation operator generalizing group-by, cross-by, and sub-by, and sub-totals," Technical report, Microsoft, Nov. 1995等を参照のこと）オペレータが提案されている。また、データ・マイニングのあるアプリケーションは、複数の集約演算の結果を用いて属性間の関係を自動的に発見し、グラフィカルな表示を行う（例えば、Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama, "Data Mining optimized association rules for numeric attributes," In Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of Database Systems, pages 182-191, June 1996, Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama, "Data mining using two-dimensional optimized association rules: Scheme, algorithms, and Visualization," In proceedings of the ACM SIGMOD conference on Management of Data, pages 13-23, June 1996. 等を参照のこと）。これらの技術では、インタラクティブに操作できることが要求されており、応答時間が重要な要素となっている。応答時間を短縮するための1つの方法として、事前に集約演算を実行しておくことが必要となる。よって、複数の集約演算を並列に高速実行することにより、OLAPやデータ・マイニングの実行を高速にすることも目的である。

【0014】さらに、複数の集約演算を実行する方法を、ハードウェアの条件や複数の集約演算の性質等に基づき、切り換えることができるようにし、様々な条件の下、複数の集約演算を常に同じ方法で実行するよりも、より高速に実行することも目的とする。

【0015】

【課題を解決するための手段】以上の目的を実現するために最も重要な技術的事項は、ディスク装置へのアクセスの回数及び時間である。今や、ディスク装置へのアクセスは、並列処理装置の各プロセッサ間の通信速度より遅い。よって、このディスク装置へのアクセスをいかに減らすかが、本発明の目的達成に大きな影響を及ぼす。以下、このディスク装置へのアクセスを減らす具体的方法について説明する。

【0016】本発明の第1の集約演算実行方法は以下の

ようなステップを含む。すなわち、(a) 各プロセッサが、自己のメモリ領域に、N個の集約演算のうちM個（N以下の整数）分の演算結果を記憶するスペースを確保するステップと、(b) 各プロセッサが、M個の集約演算をまとめて、データベースの自己の部分に対して実行するステップと、(c) 各プロセッサが、M個の集約演算の各プロセッサにおける演算結果を、集計すべきプロセッサに送信し、集計すべきプロセッサが最終結果を計算するステップと、(d) N個の集約演算の実行が終了するまで、ステップ(a)乃至(c)を繰り返すステップと、である。なお、本発明の集約演算実行方法を実施するのに適当なコンピュータ・システムは、ネットワークにより接続された複数のプロセッサが各々自己のメモリ領域及び1又は複数のグループに分けることができるデータを含むデータベースの自己の部分を使用できるよう構成されている。メモリについては、各ノードごとに分離されて用意されていても、ネットワークに接続されたメモリが、幾つかのメモリ領域に分けられており、そのメモリ領域を各プロセッサが用いる方式にしてもよい。ディスクについても同様である。

【0017】この第1の集約演算実行方法は、従来技術の欄で述べた2Pアルゴリズムをベースに考えられた方法である。この方法では、先に示したIBM SP2で、結果のグループの数が比較的小さい場合に計算速度が高速になる。

【0018】上記のステップ(b)は、(b1) 各プロセッサが、自己のメモリ領域のワークスペースに、データベースの自己の部分である部分データベースの一部を読み出すステップと、(b2) 各プロセッサが、自己のメモリ領域に格納された本ステップ以前の演算結果と読み出された部分データベースの一部とについて、M個の集約演算を実行するステップと、(b3) 各プロセッサが、部分データベースのすべてに対してM個の集約演算が実行されるまで、ステップ(b1)及び(b2)を繰り返すステップとを含むようにすることもできる。ステップ(b2)は、読み出された部分データベースの一部についてのみM個の集約演算を実行して、それからそのステップ以前の演算結果との集計を計算するようにすることもできるが、このような方法では遅くなる。

【0019】上で述べたMは、N以下の整数として、他の条件と関係なく選択することも可能であるが、集約演算の演算結果のために自己のメモリ領域に確保できるスペースから決定すると、繰り返しのステップ(ステップ(d))が少なく済むので、より高速である。

【0020】本発明の第2の集約演算実行方法は、(a) 各プロセッサが、P個の集約演算のうち自らが実行するQ個の集約演算の演算結果を記憶するスペースを、自己のメモリ領域に確保するステップと、(b) 各プロセッサが、自己のメモリ領域のワークスペースにデータベースの自己の部分である部分データベースの一部

を読み出し、読み出された部分データベースの一部をネットワークを介してブロードキャストすることを繰り返して、各プロセッサが全てのデータベースのデータに対し自らが実行するQ個の集約演算を実行するステップと、(c) P個の集約演算が実行されるまで、ステップ(a)及び(b)を繰り返すステップとを含む。第2の集約演算実行方法のためのコンピュータ・システムは、第1の集約演算実行方法と同様である。

【0021】この第2の集約演算実行方法は、BCアルゴリズムをベースに考えられた方法である。この方法では、第1の集約演算実行方法より比較的小さいグループの数が多く場合に、計算速度がより高速になる。

【0022】この第2の集約演算実行方法のステップ(a)は、(a1) 1つの集約演算の演算結果を格納できるようなスペースが1つのプロセッサの自己のメモリ領域に存在するか検査するステップと、(a2) スペースが存在する場合には、スペースを1つの集約演算の演算結果のために確保するステップと、(a3) スペースが存在しない場合には、他のプロセッサのメモリ領域に前記1つの集約演算の演算結果を格納できるようなスペースが存在するか検査するステップと、(a4) 他のプロセッサのメモリ領域にスペースが存在する場合には、当該他のプロセッサのメモリ領域のスペースを前記1つの集約演算の演算結果のために確保するステップとを含むようにすることもできる。この時、他のプロセッサのメモリ領域にスペースが存在しない場合には、後の繰り返し処理にて前記1つの集約演算を実行するようにすることもできる。どうしても、ある集約演算にスペースが存在しないような場合も生じ得るが、その場合には当該集約演算を幾つかに分けて実行するようなことも可能である。

【0023】Qについては、先のMと同様に、集約演算の演算結果のために自己のメモリ領域に確保できるスペースから決定する、ということも可能である。

【0024】さらに第2の集約演算実行方法のステップ(b)は、(b1) 各プロセッサが、自己のメモリ領域のワークスペースに、部分データベースの一部を読み出すステップと、(b2) 読み出された部分データベースの一部をネットワークを介してブロードキャストするステップと、(b3) 自己のメモリ領域に格納された本ステップ以前の演算結果と読み出された部分データベースの一部と他のプロセッサから送られてきたデータとについて、自らが実行するQ個の集約演算を実行するステップと、(b4) 各プロセッサが全てのデータベースの内容に対し自らが実行するQ個の集約演算を実行するまでステップ(b1)乃至(b3)を繰り返すステップとを含むようにすることも可能である。ステップ(b3)で、読み出された部分データベースの一部と他のプロセッサから送られてきたデータとについてQ個の集約演算を実行し、後に当該ステップ以前の演算結果と集計をとってもよいが、これでは遅くなる。

【0025】本発明の第3の集約演算実行方法は、

(a) S個の集約演算のうち実行するT個の集約演算を決定するステップと、(b) T個の集約演算を実行する際に集約される、各集約演算の各グループを取り扱うプロセッサを決定するステップと、(c) 各プロセッサが、データベースの自己の部分である部分データベースの一部を自己のメモリ領域に読み出し、読み出されたデータのうち他のプロセッサが集約すべき集約演算のグループに関するデータを当該集約演算のIDと共に他のプロセッサにネットワークを介して送信し、自己が集約すべき集約演算のグループに関するデータについてT個の集約演算を実行するステップと、(d) 各プロセッサが集約すべき集約演算のグループに関する全てのデータにT個の集約演算を実行するまで、ステップ(c)を実行するステップと、(e) S個の集約演算を実行するまで、ステップ(a)乃至(d)を繰り返すステップとを含む。第3の集約演算実行方法のためのコンピュータ・システムも、第1の集約演算実行方法と同様である。

【0026】この第3の集約演算実行方法は、Repアルゴリズムをベースに考えられた方法である。この方法では、第2の集約演算実行方法より比較的グループの数が多の場合に、計算速度がより高速になる。

【0027】第3の集約演算実行方法におけるステップ(c)は、(c1) 各プロセッサが、データベースの自己の部分である部分データベースの一部を自己のメモリ領域のワークスペースに読み出すステップと、(c2) 各プロセッサが、読み出されたデータの各部分を必要とするプロセッサを求め、当該部分を関連する集約演算のIDと共に必要とするプロセッサにネットワークを介して送信するステップと、(c3) 各プロセッサが、読み出されたデータのうち自己が集約すべき集約演算のグループに関するデータ及び他のプロセッサからのデータ及び自己のメモリ領域に格納された本ステップ以前の演算結果に対し、T個の集約演算を実行するステップとを含むようにすることも考えられる。ネットワークを介しての送信は、T個の集約演算と同時に進めてもよい。

【0028】この第3の集約演算実行方法におけるT個の集約演算の決定方法及びどのプロセッサが集約演算のどのグループを集約するかについては、各プロセッサの使用メモリ容量によって決められる。

【0029】ところで、第1乃至第3の集約演算実行方法のうち最適なアルゴリズムは、条件によって異なる。よって、プロセッサの数とデータベースの大きさとネットワークの通信速度とを含むコンピュータ・システムのパラメータ及び実行する集約演算の数と各集約演算の結果を格納するメモリの量とを含む集約演算の性質に関するパラメータを用いて、上記のうち最も高速なアルゴリズムを選び出し、最適なアルゴリズムによって複数の集約演算を実行するようにすれば、ユーザは常に高速に複数の集約演算の結果を得ることができる。なお、選択は

3つの方法の中から行わなければならないわけではなく、他の方法を含めてその中で選択することも、3つの方法のサブセットの中から選択することも可能である。

【0030】以上の述べた方法は、プログラムによって実現することも、専用の装置を用いて実現することも可能である。このように実現形態を変更することは、以下の説明を理解した当業者が通常行うことのできる事項である。

【0031】

【発明の実施の形態】初めに本発明において用いられるコンピュータ・システムのハードウェア構成について説明する。図3は、その典型的な一例を示すものである。プロセッサ・エレメントPE1(3a)、プロセッサ・エレメントPE2(3b)、及びプロセッサ・エレメントPEN(3c)がネットワーク1を介して接続されている。このプロセッサ・エレメントは図3のように3つである必要はなく、複数であればよい。一般にn個とする。プロセッサ・エレメントPE1(3a)には、メモリ5a及びディスク装置7aが接続されており、同様にプロセッサ・エレメントPE2(3b)にはメモリ5b及びディスク装置7b、プロセッサ・エレメントPEN(3c)にはメモリ5c及びディスク装置7cが接続されている。各ディスク装置には、データベースの一部分がそれぞれ記憶されている。ネットワーク1は、プロセッサ・エレメント間の通信に用いられ、高速にデータをやり取りできるようになっている。

【0032】図3は一例であって、例えば図4のように、各プロセッサ・エレメントに直接メモリが接続されておらず、メモリ9及びメモリ10がネットワーク1に接続されるような構成も可能である。この場合メモリ9は、プロセッサ・エレメントPE1用の領域9aとプロセッサ・エレメントPE2用の領域9bとに分けることができる。メモリ9はこのようにそれぞれのプロセッサ・エレメント用の領域の他、プロセッサ・エレメント間で共有する領域を設けてもよい。また、メモリ10はPEN専用となっているが、このように、ネットワーク1に各プロセッサ・エレメント用に別のメモリを接続するような構成とすることも可能である。プロセッサ・エレメント3a、3b、3cは、ディスク装置7a、7b、7cからデータを読み出すと、ネットワーク1を介して自己のメモリ領域9a、9b、10のワークエリアにデータを書き込む。

【0033】また図5のように、プロセッサ・エレメントとメモリは図3と同様な接続をしているが、ディスク装置がネットワーク1に接続されているような構成も考えられる。ここで、ディスク装置11はプロセッサ・エレメントPE1用のデータ11aとプロセッサ・エレメントPE2用のデータ11bとを有している。ディスク装置11は、プロセッサ・エレメントPE1及びPE2用データの他、共有のデータ等を含むことができる。ま

た、ディスク装置 13 は、プロセッサ・エレメント PEn 用のデータを格納している。このように、各プロセッサ・エレメント専用のディスク装置をネットワーク 1 に別々接続する構成も可能である。これらディスク装置内のデータは、ネットワーク 1 を介してプロセッサ・エレメント 3a, 3b, 3c によって読み出され、メモリ 5a, 5b, 5c のワークエリアに取り込まれる。

【0034】図 3、図 4、及び図 5 を幾つか組み合わせた構成も可能である。例えば、幾つかのプロセッサ・エレメントは図 3 のような構成をとっており、幾つかのプロセッサは図 4 や図 5 のような構成をすることも可能である。また、図 3、図 4、図 5 以外の構成も可能であり、要するに、プロセッサ間でメモリやディスク装置を共有する必要はない。

【0035】以上のようなハードウェア構成を前提として、以下どのような処理を行うかを説明する。なお、以下に説明する処理では、システム全体のための処理も一部に含まれるが、その処理は複数のプロセッサ・エレメントのうち 1 つをその処理のために割当てたり、そのような処理専用のプロセッサを用意してもよい。

【0036】1. 2Pm アルゴリズム (図 6)
本アルゴリズムは、従来技術で説明した 2P アルゴリズムをベースとしたものであり、複数の集約演算を並列に実施するように拡張したものであるため、2Pm アルゴリズムと呼ぶことにする。複数の集約演算を実施する際に、ディスク装置へのアクセスを最小限にすることで高

速化する。なお、全部で N 個の集約演算を実行することを前提とする。

【0037】(1) 最初に、自己のメモリ領域に M 個の集約演算の演算結果を記憶するスペースを確保する (ステップ 210)。この M 個はその演算結果を各プロセッサ・エレメントのメモリ領域に格納できる範囲で決められるが、なるべくたくさんの集約演算を 1 度に行う方がディスク・アクセスを共有することができるので、処理が高速になる。また、本アルゴリズムを実行する全てのプロセッサ・エレメントが、全て同じ M 個の集約演算を実行する方が高速処理できる。これは、後に説明する集計処理が必要となるため、プロセッサ・エレメント毎に異なる個数の集約演算を実施していると、集計処理ができない演算結果の一部がメモリに残ってしまい、他のプロセッサ・エレメントで演算の実行が終了するのを待たなければならないといった問題が生じるからである。よって、実行する集約演算は、本アルゴリズムを実行するプロセッサ・エレメントが同じ個数の同じ集約演算を 1 度に行うことができるという観点から選択すると、処理の高速化に効果的である。なお、演算結果を記憶するスペースを演算を実行する前に知らなければならないが、これは少しのデータに対して試算をしてみれば、演算結果の規模は把握することができ、大きな負担ではない。

【0038】以下に、ステップ 210 の擬似コードを示しておく。

【表 3】

```
// 変数
N : number of node
mem : size of memory
MEM : variable ( 0 ≤ MEM ≤ mem )
q : query
NL : list of queries
tmpNL : list of queries # 次回以降に実行する分
ML : list of queries # 今回実行する分

// 初期化
NL = { q1, q2, ..., qi, ..., qQ } # list of Q queries.
mem = getMaxMemSize() # メモリサイズを得る。
# 各ノードが協調して最も小さいサイズに合わせる

# 各ノードが以下を実行
while NL ≠ NIL 且つ tmpNL ≠ NIL do # NIL : 空リスト
begin
MEM = mem :
tmpNL = NIL :
NL = NIL :
while NL ≠ NIL do # NL が 空になるまで実行
begin

q = pop( NL ) : # pop() リストの先頭を取ってくる
if ( MEM > |q| ) then # q のメモリが確保できるか調べる
begin # |q| は演算 q が使用するメモリサイズ
```

```

append( ML, q ) :      # ML に q を加える
MEM = MEM - |q| :      # 残りのメモリサイズから |q| を引く
end
else
begin
append( tmpML, q ) :   # 演算 q を未処理リストに加える
end
end
# ML 中の query について集約演算を実行
# tmpML は、次回以降に
while ML ≠ NIL do      # ML が NIL になるまで実行
begin
q = pop( ML ) :        # ML から先頭の演算を取ってくる
allocate ( q ) :       # 演算 q 用のメモリを確保
end
2Pm ( ) :              # ML のリストにあった query について
                        # 2Pm アルゴリズムを実行
copy( ML, tmpML ) :    # ML に、tmpML をコピー
end

```

【0039】(2) データベースの自己の部分に対して M 個の集約演算を実行する(ステップ220)。この処理をより詳しく説明すると、最初に自己のメモリ領域のワークスペースに、データベースの自己の部分のうち、今回処理する分を読み出す。そして、自己のメモリ領域に格納された、これまでの演算結果(一番最初のステップにおいては存在しない)と、データベースの自己の部分のうち、読み出された今回処理する分とについて、M 個の集約演算を実行する。この2つのステップを、データベースの自己の部分すべてに対して実行するまで繰り返す。具体例については後に説明する。1回読み出したデータに対し、従来の方法では1つの集約演算しか行われていなかったため、そのまま M 個の集約演算を実行する場合に適用すると、M 回データを読み出さなければならなかった。一方、本発明では、1回読み出したデータに対し、M 個の集約演算が実施されているため、ディスク・アクセスのための時間は全体として $1/M$ となる。

【0040】(3) そして、M 個の集約演算の演算結果を集計担当プロセッサ・エレメント(集計担当ノードともいう)に送信し、集計担当プロセッサ・エレメントが最終結果を計算する(ステップ230)。集計担当プロセッサ・エレメントは、1つでも複数でもよい。例えば、M 個のプロセッサ・エレメントが共に本アルゴリズムを実行している場合には、各プロセッサが1つの集約演算の演算結果の集計担当プロセッサ・エレメントとし、最終結果を計算するようにすることもできる。M 個よりプロセッサ・エレメントが少ない場合には、1つのプロセッサ・エレメントが複数個の集約演算の演算結果を集計してもよいし、ただ1つのプロセッサ・エレメントが M 個の集約演算の演算結果をすべて集計するように

してもよい。どのプロセッサ・エレメントがどの集約演算の演算結果を集計するかは、予め決めておく。

【0041】また、集約演算の演算結果における各グループをプロセッサ・エレメントごとに割当て、集計する方法も考えられる。例えば、プロセッサ・エレメント1が集約演算1のグループ1及び2を集計し、プロセッサ・エレメント2が集約演算1のグループ3及び4と集約演算2のグループ1及び2を集計する、といった具合である。

【0042】(4) M 個の集約演算の最終結果が得られると、N 個の集約演算が全て実行されたか判断される(ステップ240)。もし、N 個の集約演算がすべて実行されたわけではない場合には、ステップ210に戻って処理を繰り返す。この場合、通常はもとの M と異なる M で繰り返しを実行する。もし、N 個の集約演算がすべて実行された場合には、処理を終了する。

【0043】では実際の実行例を図7乃至図10の例を用いて説明する。図7はプロセッサ・エレメント1用のディスク装置に記憶されたリレーションを、図8はプロセッサ・エレメント2用のディスク装置に記憶されたリレーションを、図9はプロセッサ・エレメント3用のディスク装置に記憶されたリレーションを、図10はプロセッサ・エレメント4用のディスク装置に記憶されたリレーションを、それぞれ表し、コンピュータ・システムには4つのプロセッサ・エレメントが存在するとする。表の各列は、右から売上げの月、日、曜日、場所、店番号、製品番号、売上げ数をそれぞれ表す。表の各行は、売上げが発生した時に生成されるとする。

【0044】月ごとに売上げ数を合計する演算(演算1)と、曜日ごとに売上げ数を合計する演算(演算2)

と、場所ごとに売上げ数を合計する演算（演算3）と、製品番号ごとに売上げ数を合計する演算（演算4）と、場所及び曜日ごとに売上げ数を合計する演算（演算5）とを一度に実行することとする。

【0045】プロセッサ・エレメント1は、図7の第1行目のデータをディスク装置からメモリのワークスペースに読み出す。そして、演算1を実施するため、月と売上げ数の列を参照する。月は“Apr”で売上げ数“3”が最初に演算1の結果として記憶される。次に、演算2を実行するため、曜日と売上げ数の列を参照し、曜日は“Sun”で売上げが“3”が演算2の結果として記憶される。演算3を実行するために、場所と売上げ数の列を参照し、場所“Kyuushuu”で売上げ数“3”が演算3の結果として記憶される。演算4を実行するために、製品番号と売上げ数の列を参照し、製品番号2で売上げ数“3”が演算4の結果として記憶される。演算5を実行するために、場所、曜日、売上げ数の列が参照され、場所及び曜日が“Kyuushuu”“Sun”で売上げ“3”が演算5の結果として記憶される。

【0046】次にプロセッサ・エレメント1は、図7の第2行目のデータをディスク装置からメモリのワークスペースに読み出す。そして、演算1を実行するため、月と売上げ数の列を参照する。ここで、読み出したデータの月のデータは、先に記憶した最初の演算1の結果の月とは異なるため、新たに月は“Aug”で売上げ“4”を演算1の結果に追加する。次に、演算2を実行するため、曜日と売上げ数の列を参照する。ここで、読み出したデータの曜日のデータとこれまでの演算2の実行結果の曜日のデータとは同じなので、演算2の結果の曜日“Sun”の項目に売上げ“4”を足して“7”を記憶する。演算3を実行するため、場所と売上げ数の列を参照する。ここで、読み出したデータの場所のデータは、先に記憶した、これまでの演算3の結果の場所とは異なるため、新たに場所は“Hokkaido”で売上げ数“4”を演算3の結果に追加する。演算4を実行するため、製品番号と売上げ数の列を参照する。ここで読み出したデータの製品番号のデータは先に記憶した、これまでの演算4の結果の製品番号とは異なるので、新たに製品番号“1”で売上げ“4”を演算4の結果に追加する。演算5を実行するため、場所及び曜日と売上げ数の列を参照する。ここで読み出したデータの場所及び曜日は、先に記憶した、これまでの演算5の結果の場所及び曜日とは異なるため、新たに場所及び曜日が“Hokkaido”“4”を演算5の結果に追加する。

【0047】次にプロセッサ・エレメント1は、図7の第3行目のデータをディスク装置からメモリのワークスペースに読み出す。演算1を実行するため、月と売上げ数の列を参照する。ここで、読み出したデータの月のデータは、これまでの演算1の結果の月とは異なるため、新たに月は“Jun”で売上げ“2”を演算1の結果に追加

する。演算2を実行するため、曜日と売上げ数の列を参照する。ここで、読み出したデータの曜日はこれまでの演算2の結果の曜日とは異なるため、新たに曜日は“Mon”で売上げ数“2”を追加する。演算3を実行するために、場所と売上げ数の列を参照する。読み出したデータの場所とこれまでの演算3の結果の場所とは一致するものがある。よって、場所“Hokkaido”で売上げ数“4”の項目に売上げ数“2”を足して“6”を記憶する。演算4を実行するために、製品番号と売上げ数の列を参照する。読み出したデータの製品番号のデータと、これまでの演算4の結果の製品番号とは一致するものがある。よって、製品番号“1”で売上げ数“4”の項目に売上げ数“2”を足して“6”を記憶する。演算5を実行するために、場所及び曜日と売上げ数の列を参照する。ここで、読み出したデータの場所及び曜日のデータは、これまでの演算5の結果の場所及び曜日とは異なるため、新たに場所及び曜日“Hokkaido”“Mon”で売上げ数“2”を追加する。

【0048】ここまでの処理で、プロセッサ・エレメント1のメモリには図11の結果が記憶される。このような処理をプロセッサ・エレメント1のディスク装置に記憶された図7のリレーションに対し繰り返すと、プロセッサ・エレメント1のメモリには図12のような結果が記憶されることになる。

【0049】プロセッサ・エレメント2、プロセッサ・エレメント3、及びプロセッサ・エレメント4でも同様の処理が実施され、ディスク装置に記憶された図8、図9、及び図10のリレーションに対する演算1乃至演算5の結果は、それぞれ図13、図14、及び図15に表わされるようになる。

【0050】そして、集計処理を実施すれば、集約演算1乃至5の最終結果を得ることができる。本例では、各プロセッサ・エレメントが集約演算1乃至5の一部のグループについて集計するという集計方法を採用している。すなわち、プロセッサ・エレメント1は、演算1の月“Jan”と“Feb”のグループ、演算2の曜日“Mon”と“Tue”のグループ、演算3の場所“Hokkaido”のグループ、演算4の製品番号“1”のグループ、演算5の場所及び曜日のうち場所“Hokkaido”のグループについて集計する。プロセッサ・エレメント2は、演算1の月“Mar”と“Apr”のグループ、演算2の曜日“Wed”と“Thu”のグループ、演算3の場所“Kanto”のグループ、演算4の製品番号“2”のグループ、演算5の場所及び曜日のうち場所“kanto”のグループについて集計する。

【0051】さらに、プロセッサ・エレメント3は、演算1の月“May”と“Jun”のグループ、演算2の曜日“Fri”と“Sat”のグループ、演算3の場所“Kansai”のグループ、演算4の製品番号“1”のグループ、演算5の場所及び曜日のうち場所“Kansai”の

グループについて集計する。プロセッサ・エレメント4は、演算1の月“Jul”と“Aug”のグループ、演算2の曜日“Sun”のグループ、演算3の場所“Kyuushuu”のグループ、演算5の場所及び曜日のうち場所“Kyuushuu”のグループについて集計する。

【0052】よって、各プロセッサ・エレメントは、自己が保持しているグループのデータのうち、自己が集計するグループのデータ以外のデータを集計担当ノードに送信する必要がある。

【0053】以上、プロセッサ・エレメント1が集計した結果を図16、プロセッサ・エレメント2が集計した結果を図17、プロセッサ・エレメント3が集計した結果を図18、プロセッサ・エレメント4が集計した結果を図19に示す。

【0054】2. BCm アルゴリズム (図20)
本アルゴリズムは、従来技術で説明したBCアルゴリズムをベースとしたものであり、複数の集約演算を並列に実行するように拡張したものであるため、BCmアルゴリズムと呼ぶことにする。本アルゴリズムも、複数の集約演算を実行する際に、ディスク装置へのアクセスを最小限にすることで高速化する。なお、全部でP個の集約演算を実行することを前提とする。

【0055】(1) まず、自らが実行するQ個の集約演算の演算結果を記憶するスペースを自己のメモリ領域に確保する(ステップ310)。このQ個は、2Pmアルゴリズムで説明したM個と同様に、各プロセッサ・エレメントのメモリ領域に格納できる範囲で決められるが、なるべくたくさんの集約演算を一度に実行する方がディスク装置へのアクセスを共有することになるので、処理が高速になる。なお、Q個の集約演算は、各プロセッサ・エレメントで異なる集約演算を実行するようにしてもよいし、各プロセッサ・エレメントで同一にしておき、集約演算の結果のグループを分けるということも可能である。

【0056】どの集約演算をどのプロセッサ・エレメントに割当てるとかという処理の一例を図21に示す。最初に未実行の集約演算を1つ選択する(ステップ410)。そして、プロセッサ・エレメントを1つ選択する(ステップ420)。その後、選択されたプロセッサ・エレメントのメモリに選択された集約演算の結果を記憶

するスペースがあるか検査する(ステップ430)。もし、そのプロセッサ・エレメントのメモリにそのスペースがあるならば、そのプロセッサ・エレメントに選択された集約演算を割当てる(ステップ440)。一方、選択されたプロセッサ・エレメントのメモリにその集約演算の演算結果を格納するスペースがない場合には、他のプロセッサ・エレメントのメモリについて検査する(ステップ450)。但し、ある集約演算についてすべてのプロセッサ・エレメントについて検査したにもかかわらず、その演算結果用のスペースが確保できなかった場合には(ステップ460)、当該集約演算は別途処理する(ステップ470)。別途処理というのは、図20のステップ350で繰り返し処理を実行するようになるので、後の繰り返し処理の時に実行するようにするか、場合によっては、当該集約演算を幾つかのグループに分割して実行するように変更する等の処理を含む。

【0057】そして、他の集約演算についてプロセッサ・エレメントの割当てを決定するわけであるが、全てのプロセッサ・エレメントのメモリに既に十分なスペースがない場合は図20のステップ350による繰り返し処理の際に再度割当てを行うので、又、全ての未実行の集約演算について割当て処理を実行してしまった場合(ステップ480)には、割当ての済んだ集約演算を実行するため、この処理は一旦終了する(ステップ490)。

【0058】集約演算のプロセッサ・エレメントへの上記の割当て処理は一例であって、他の方法によって割当てすることもできる。ユーザが、意図的にある集約演算を特定のプロセッサ・エレメントに割当てることができるようにすることも可能である。なお、先にも述べたが、各プロセッサ・エレメントで同一集約演算を実行するが、各プロセッサ・エレメントで集約するグループを変えることもできる。この場合には、同一種類の集約演算であるがグループが異なるため別集約演算として図21のプロセスにてプロセッサ・エレメントを割当てるともできる。また、このような実行方法のために、別の割当てアルゴリズムを用意することも可能である。

【0059】図21をより詳しくした擬似コードを以下に示す。

【表4】

```
// 変数
N      : number of node
m_i    : memory size for each node (1 ≤ i ≤ N)
q      : query
q_i    : query (1 ≤ i ≤ Q)
QL     : list of queries to be aggregated # 集約すべき演算のリスト
tmpQL  : list of queries # 次回以降実行する分
ML     : list of queries # 今回実行する分
// 初期化
QL     = { q_1, q_2, ..., q_i, ..., q_Q } # list of Q queries.
```

```

while QL ≠ NIL 且つ tmpQL ≠ NIL do    # NIL : 空リスト
begin
  for i=0 to N do
    begin
      m_j = getMaxMemSize( j ) ; # 各ノードのメモリサイズを得る。
    end
    tmpQL = NIL ;
    ML = NIL ;
    while QL ≠ NIL do    # QL が 空になるまで実行
      begin

        q = pop( QL ) ;    # pop() リストの先頭を取ってくる
        for j=1 to N do
          begin

            if ( m_j > |q| ) then # q のメモリが確保できるか調べる
              begin
                # |q| は演算 q が使用するメモリサイズ

                append( ML, q ) ;    # ML に q を加える
                m_j = m_j - |q| ;    # 残りのメモリサイズから |q| を引く
                break ;    # for 文を抜ける
              end
            end
          if (q の割り当て先が見つからない) then
            begin
              append( tmpQL, q ) ; # 演算 q を未処理リストに加える
            end
          end
        # ML 中の query について集約演算を実行
        # tmpQL は、次回以降に
        while ML ≠ NIL do    # ML が NIL になるまで実行
          begin
            q = pop( ML ) ;    # ML から先頭の演算を取ってくる
            allocate ( q ) ;    # 演算 q 用のメモリを確保
          end
          BQm( ) ;    # ML のリストにあった query について
                    # BQmアルゴリズムを実行
          copy( QL, tmpQL ) ;    # QL に、tmpQL をコピー
        end
      end
    end
  end

```

【0060】(2) 図20に戻って、データベースの自己の部分のうち、今回処理する分を読み出し、読み出したデータをネットワークを介してブロードキャストする(ステップ320)。ブロードキャストであるから、同じ内容のデータが全てのプロセッサ・エレメントに送信される。この際、どのデータがどのプロセッサ・エレメントで必要となるかは判断しなくともよい。先に述べたRepアルゴリズム及び後に述べるRepMアルゴリズムにおいては、どのプロセッサがどのグループについて集約を行っているか又はどのプロセッサがどの集約演算

のどのグループについて集約を行っているか判断して、必要なプロセッサ・エレメントに送信するようになっていたが、本アルゴリズムにおいては、全てのプロセッサ・エレメントが同一集約演算を実行しているわけではないからである。全プロセッサ・エレメントで用いないようなデータが存在することが予め分かっている場合には、そのデータの送信を省略できる。

【0061】(3) そして、他のプロセッサ・エレメントからのデータと、自ら読み出したデータと、これまでの演算結果(一番最初の場合には存在しない)に対し

て、Q個の集約演算を実行する(ステップ330)。

(4) この(2)(ステップ320)(3)(ステップ330)の処理を、全てのデータベースのデータについてQ個の集約演算を実行し終わるまで繰り返す(ステップ340)。個々のプロセッサ・エレメント用のデータベースの大きさにばらつきがある場合には、自己のデータベースを全て処理しても他のプロセッサ・エレメントからデータが送信されてきたり、逆に他のプロセッサ・エレメントからは送られてこないが自己のデータベースを全て処理していないという場合も考えられる。この場合、ステップ330は、自己のデータベース又は他プロセッサ・エレメントからの未処理のデータとそれまでの演算結果とに対しQ個の集約演算を実行する、ということになる。

【0062】(5)全てのデータベースのデータに対してQ個の集約演算を実行した場合には、P個の集約演算すべてについて実行が終了したか判断する(ステップ350)。終了していない場合には、別の集約演算に対してステップ310以下を実行する。終了している場合には、全処理を終了する(ステップ360)。

【0063】このBCmアルゴリズムでは、ある集約演算を実行するプロセッサ・エレメントが、全てのデータについて演算を実行するが、ディスク装置へのアクセスは、自己のディスク装置に対してのみなので、このディスク装置へのアクセスのコストは削減されている。また、複数の集約演算でこのディスク装置へのアクセスを共有するので、全体としてコストは削減されている。

【0064】以上の処理を図7、図8、図9、図10を用いて具体的に説明する。ここでは、プロセッサ・エレメント1が先に示した演算1を、プロセッサ・エレメント2が演算2及び3を、プロセッサ・エレメント3が演算4を、プロセッサ・エレメント4が演算5を実行するものとする。プロセッサ・エレメント2の動作を説明する。なお、1行ずつ読み出すような説明であるが、任意の行数ずつ読み出すようにすることも可能である。

【0065】プロセッサ・エレメント2は、図8の第1行目をメモリのワークスペースに読み出す。そして、ネットワークを介して全プロセッサ・エレメントに送信する。他のプロセッサ・エレメントも同様に読み出したデータ(各データベースの第1行目)を全プロセッサ・エレメントに送信するので、プロセッサ・エレメント2のメモリのワークスペースには、図7、図8、図9及び図10の第1行目が格納されている。そして、先に示した演算2及び演算3を実行する。すなわち、曜日ごとに売上げ数を合計する演算(演算2)と、場所ごとに売上げ数を合計する演算(演算3)である。

【0066】プロセッサ・エレメント2がワークスペースに存在するデータに対し演算2を実行すると、曜日“Sat”で売上げ数“7”、曜日“Wed”で売上げ数“15”、曜日“Sun”で売上げ数“3”が得られる。また、

演算3を実行すると、場所“Kyuushuu”で売上げ数“10”、場所“Kansai”で売上げ数“6”、場所“Hokkaido”で売上げ数“9”が得られる。

【0067】この後、プロセッサ・エレメント2は、図8の第2行目をメモリのワークスペースに読み出す。そして、ネットワークを介して全プロセッサ・エレメントに送信する。他のプロセッサ・エレメントも同様に読み出したデータ(各データベースの第2行目)を全プロセッサ・エレメントに送信するので、プロセッサ・エレメント2のメモリのワークスペースには、図7、図8、図9及び図10の第2行目が格納されている。

【0068】プロセッサ・エレメント2がワークスペースのデータ及びこれまでの演算結果に対して演算2を実行すると、曜日“Sat”で売上げ数“16”、曜日“Wed”で売上げ数“15”、曜日“Sun”で売上げ数“7”、曜日“Tue”で売上げ数“8”、曜日“Thu”で売上げ数“1”が得られる。また、同様に演算3を実行すると、場所“Kyuushuu”で売上げ数“11”、場所“Kansai”で売上げ数“6”、場所“Hokkaido”で売上げ数“21”、場所“Kanto”で売上げ数“9”が得られる。

【0069】さらに、プロセッサ・エレメント2は、図8の第3行目をメモリのワークスペースに読み出す。そして、ネットワークを介して全プロセッサ・エレメントに送信する。他のプロセッサ・エレメントも同様に読み出したデータ(各データベースの第3行目)を全プロセッサ・エレメントに送信するので、プロセッサ・エレメント2のメモリのワークスペースには、図7、図8、図9及び図10の第3行目が格納されている。

【0070】プロセッサ・エレメント2がワークスペースのデータ及びこれまでの演算結果に対して演算2を実行すると、曜日“Sat”で売上げ数“16”、曜日“Wed”で売上げ数“15”、曜日“Sun”で売上げ数“15”、曜日“Tue”で売上げ数“8”、曜日“Thu”で売上げ数“9”、曜日“Mon”で売上げ数“4”が得られる。また、同様に演算3を実行すると、場所“Kyuushuu”で売上げ数“11”、場所“Kansai”で売上げ数“6”、場所“Hokkaido”で売上げ数“31”、場所“Kanto”で売上げ数“19”が得られる。

【0071】以上のような処理を繰り返し、図7、図8、図9及び図10の全てのデータに対して、プロセッサ・エレメント2が演算2及び演算3を実行すると、図22のようなデータが、プロセッサ・エレメント2のメモ리에記憶されることになる。なお、プロセッサ・エレメント1、プロセッサ・エレメント3、プロセッサ・エレメント4の最終実行結果を図23に示しておく。

【0072】3. Repmアルゴリズム(図24)

本アルゴリズムは、従来技術の欄で説明したRepアルゴリズムをベースとしたものであり、複数の集約演算を並列に実施するように拡張したものであるため、Rep

mアルゴリズムと呼ぶことにする。複数の集約演算を実施する際に、ディスク装置へのアクセスを最小限にすることで全体を高速化する。なお、全部でS個の集約演算を実行するものとする。

【0073】(1) 最初に、S個の集約演算のうちどの集約演算を実行するか決定する(ステップ510)。ここで実行される集約演算の個数をT個とする。以下の説明では、同時に実行する集約演算は、各プロセッサについて同一にし、その代わりにグループの種類を変えるよ

うにする。但し、必ず各プロセッサで実行する集約演算が同一である必要はない。また、このステップは、システム全体で使用可能なメモリ容量を勘案して実施されるべきであるが、必ずしもメモリ容量のみにより制限する必要はなく、ユーザがメモリ容量の限度において指定するようにしてもよい。以下に、本ステップの実施方法の一例を擬似コードによって示す。

【0074】

【表5】

```

入力: 集約演算の個数: s
      各集約演算の結果の大きさの見積もり: n(1), n(2), ..., n(s)
      システム全体の主記憶の大きさ: M

出力: メモリに入る集約演算を区切って出力する。

アルゴリズム:
R = { 1, 2, ..., s };           # 処理していない集約演算の集合
while R is not empty do
  m = M;                         # 残りのメモリサイズ
  Q = empty;                     # 結果が入る変数Qを空に初期化
  while m >= 0 or R is empty do
    x = an element in R;         # R から一つ要素を取り出して
    R = R - { x };               # R から取り除き、
    Q = Q + { x };               # Q に加える。
    m = m - n(x);               # その要素が使用するメモリを引く。
  done:
  output(Q);                     # Qをメモリに入る集約演算のグループとして
                                # 出力する。
done.

```

【0075】(2) 次に、どの集約演算のどのグループをどのプロセッサが実行するか決定する(ステップ520)。例えば、Aという演算のグループとしてb, c, d, eが存在するならば、そのb, c, d, eをそれぞれプロセッサ1, 2, 3, 4に割り当て、Bという演算のグループとしてf, g, h, iが存在するならば、そのf, g, h, iをそれぞれプロセッサ1, 2, 3, 4に割り当てる等の処理を実施する。なお、各集約演算にどのようなグループが存在するかが予め分からないと本ステップは実行できないが、小規模のサンプリングを実行すれば、グループの種類を把握することができる。また、どのようにグループを割り当てるかは、様々な方法が考えられる。例えば、グループを入力とするハッシュ関数を用意しておき、ハッシュ値の範囲によってプロセッサを割り当てることも可能であるし、ユーザ指定により行うことも可能である。

【0076】以上のステップ510及び520は、システム全体として実施する必要がある。よって、この処理は、特定のプロセッサを1つ割り当てて実施することも可能であるし、システムに制御用のプロセッサを設け、そのプロセッサに実行させることも可能である。

【0077】(3) そして、各プロセッサはデータベー

スの自己の部分から今回処理する分を読み出す(ステップ530)。例えば、データベースの1タブルを読み出すようにすることも、複数のタブルを読み出すようにすることも可能である。

(4) 各プロセッサは、読み出したデータのうち、実行する集約演算に必要なデータを選択する(ステップ540)。例えばデータベースの1タブルを読み出した場合、集約演算によっては不必要な属性についてのデータも読み出していることになるので、システム全体として不必要なデータについては破棄する。ここで、集約演算はステップ510で決定されたT個の集約演算であるから、このT個の集約演算に関係のないデータが破棄される。従来のRepアルゴリズムでは、1つの集約演算に関係するデータのみ用いられていたため、Repmアルゴリズムではディスク装置への1回のアクセスが従来のT倍有効利用されていることが分かる。

【0078】(5) 次に、各プロセッサは、読み出したデータのうち他のプロセッサが集約すべきデータを、関係する集約演算のIDと共に当該他のプロセッサにネットワーク1を介して送信する(ステップ550)。実行する集約演算によって必要とするデータの属性が決定され、その属性値によりグループが決定されるので、この

属性値を入力とするハッシュ関数を用意し、その出力値により送信先のプロセッサが判別できるようにする。T個の集約演算の各々について本処理を実施する。この際、データを受信するプロセッサがどの集約演算についてのデータが送られてきたのが判別できるように集約演算のIDを付して送信する。

(6) 各プロセッサは、そのプロセッサが集約すべき集約演算のグループに関するデータについてT個の集約演算を実行する(ステップ560)。この処理は、本ステップを実施するまでの演算結果と、読み出したデータのうち当該プロセッサが必要とするデータと、他のプロセッサから送られてきたデータとについて実施される。

【0079】(7)そして、各プロセッサが、集約すべき集約演算のグループに関する全てのデータについてT個の集約演算を実行するまで、ステップ530乃至560を繰り返す(ステップ570)。すなわち、自己が読み出したデータ及び受信したデータが、全データベースの必要な部分となるまで繰り返される。

(8)もし、この処理が終了すると、T個の集約演算の処理が終了したことになる。よって、次のT個(これまでのTとは異なる場合もある)の集約演算についてステップ510以下を、全ての(S個)の集約演算についての処理を実行するまで行う(ステップ580)。

【0080】では、先に示した図7乃至図10の例を用いて実行例を説明する。この際、同時に実行される集約演算は、先に示した演算1(月ごとの売上げ数を合計する演算)及び演算2(曜日ごとの売上げ数を合計する演算)とする。プロセッサ・エレメントは4つである。そして、プロセッサ・エレメント1は、演算1のJan及びFebのグループ、演算2のMon及びTueのグループを集約する。プロセッサ・エレメント2は、演算1のMar及びAprのグループ、演算2のWed及びThuのグループを集約する。プロセッサ・エレメント3は、演算1のMay及びJunのグループ、演算2のFri及びSatのグループを集約する。プロセッサ・エレメント4は、演算1のJul及びAugのグループ、演算2のSunのグループを集約する。

【0081】プロセッサ・エレメント3の処理を代表して説明する。図25はプロセッサ・エレメント3が図9に示したプロセッサ・エレメント3用のデータベースの第1行を読み出した状態を示す。この例では1行ずつ読み出すが、一度に読み出す行数は任意である。本例では、日にち(day)の列、場所(location)の列、店番号(shop#)の列及び製品番号(product#)の列は、演算1及び2に必要でないので破棄される(ステップ540参照)。一方、月(month)の列と週(week)の列と売上げ(sold)の列は演算1及び2に必要である。そして、第1行目の月"Aug"及び売上げ"7"は、演算1のIDと共に集約を担当するプロセッサ・エレメント4に送信される(ス

テップ550参照)。また、第1行目の週"Sat"及び売上げ"7"は、読み出したプロセッサ・エレメント3が行う演算2の集約すべきグループであるから、自ら集約を行う。ここでは最初のデータであるからそのまま記録する。プロセッサ・エレメント1及び2及び4も同時に、自己のデータベースの第1行目を読み出して処理を行っているとなると、プロセッサ・エレメント3には何も送られてこない。

【0082】次に、プロセッサ・エレメント3が図9の第2行目を読み出した状態(不要な部分を除く)を図26に示す。月"Apr"及び売上げ"8"はプロセッサ・エレメント2に演算1のIDと共に送信される。また、週"Tue"及び売上げ"8"はプロセッサ・エレメント1に演算2のIDと共に送信される。プロセッサ・エレメント1及び2及び4も同時に、自己のデータベースの第2行目を読み出して処理を行っているとなると、プロセッサ・エレメント4から"Sat"及び"9"が演算2のIDと共に送信されてくるので、プロセッサ・エレメント3は、第1行目の結果を参照して、"Sat"及び"16"を得る。

【0083】プロセッサ・エレメント3が図9の第3行目を読み出した状態(不要な部分を除く)を図27に示す。月"May"及び売上げ"8"は、プロセッサ・エレメント3が集約すべき演算1のグループであるので、ここでは"May"及び"8"をそのまま記録する。週"Thu"及び売上げ"8"は、プロセッサ・エレメント2に演算2のIDと共に送信される。プロセッサ・エレメント1及び2及び4も同時に、自己のデータベースの第3行目を読み出して処理を行っているとなると、プロセッサ・エレメント1からは演算1のIDと共に"Jun"及び"2"、プロセッサ・エレメント2からは演算1のIDと共に"May"及び"2"が送られてくるので、プロセッサ・エレメント3はそれまでの結果を参照して、"May"及び"10"、"Jun"及び"2"を格納する。なお、ここでは、自らが読み出したデータと送信されてきたデータを別個に集約しているように説明しているが、これは説明の都合上であって、区別なく集約することができる。

【0084】プロセッサ・エレメント3が図9の第4行目を読み出した状態(不要な部分を除く)を図28に示す。月"Mar"及び売上げ"8"は、プロセッサ・エレメント2が集約すべき演算1のグループであるので、プロセッサ・エレメント2に演算1のIDと共に送信する。また、週"Fri"及び売上げ"8"は、プロセッサ・エレメント3が集約すべき演算2のグループであるので、ここではそのまま格納する。プロセッサ・エレメント1及び2及び4も同時に、自己のデータベースの第4行目を読み出して処理を行っているとなると、プロセッサ・エレメント1からは演算1のIDと共に"May"及び"4"と演算2のIDと共に"Fri"及び"4"が、プロセッサ・エレメント2からは演算2のIDと共に"Fri"及

び"9"が、プロセッサ・エレメント4からは演算1のIDと共に"Jun"及び"3"と演算2のIDと共に"Set"及び"3"が送られてくる。よって、プロセッサ・エレメント3は、これらのデータから、"May"及び"14"と"Jun"及び"5"、"Fri"及び"21"と"Set"及び"19"を計算する。

【0085】以下これらの演算を繰り返すと、プロセッサ・エレメント3においては図18の演算1及び演算2の部分が得られる。同様にして、プロセッサ・エレメント1においては図16の演算1及び演算2の部分、プロセッサ・エレメント2においては図17の演算1及び演算2の部分、プロセッサ・エレメント4においては図19の演算1及び演算2の部分が計算されることとなる。なお、図29に演算1のIDが付されて各プロセッサ・エレメントから他のプロセッサ・エレメントに送信されるデータの宛先をまとめておく。逆に、演算1について、各プロセッサ・エレメントに送られてくるデータの送信元を図30にまとめておく。さらに、図31に演算2のIDが付されて各プロセッサ・エレメントから他のプロセッサ・エレメントに送信されるデータの宛先をまとめておく。逆に、演算2について、各プロセッサ・エレメントに送られてくるデータの送信元を図32にまとめておく。

【0086】これまで説明してきたアルゴリズムをまとめてみる。2Pmアルゴリズムは、各プロセッサ・エレメントが自己のディスク装置のデータに対して集約演算を実行するため、集約演算の最終結果と同じサイズのメモリを各プロセッサ・エレメントがそれぞれ用意しなければならない。しかし、通信コストが少ない分、最終結

果が小さい場合には、他のアルゴリズムに比べて十分速い。

【0087】BCmアルゴリズムは、集約演算の最終結果を全プロセッサ・エレメントのメモリで分担できるため2Pmアルゴリズムが一度に処理できる集約演算のプロセッサ数倍だけ多くの集約演算に対してディスク装置へのアクセスを共有することができる。最終結果が1プロセッサ・エレメントのメモリに入りきらない場合、2Pmアルゴリズムはメモリに入りきらない分を何度か繰り返し処理を行う必要があるが、BCmアルゴリズムではメモリ内に収まるようにできるため、一度に処理することができ、このような場合には2Pmアルゴリズムより高速に処理することができる。

【0088】最終結果を格納するのに必要なメモリサイズが非常に大きくなった場合、Rep mアルゴリズム以外のアルゴリズムはディスク装置へのアクセスや集計のコストが逆に大きくなるため、Rep mアルゴリズムが有利になる場合もある。

【0089】よって、常に最適なアルゴリズムは同一ではないので、Rep mアルゴリズム、2Pmアルゴリズム及びBCmアルゴリズムのうち最適なアルゴリズムを選択し、そのアルゴリズムによって複数の集約演算を実行することが好ましい。そこで、各アルゴリズムを実行する際のコストを見積もることとする。以下、コストモデルについて説明する。

【0090】最初に、生じ得る各コストを表6に掲げる。

【表6】

記号	内 容	値
N	ノード数	可変
R	リレーションの大きさ (Byte)	5G
R	Rの行数	10000000
R ₁	ノード1の行数	R /N
R _p	リパーティション後の1ノード当りの行数	$\max(R_1 , 1/S/Q)$
P	ページ・サイズ	4KB
Q	集約演算の数	可変
p	プロジェクトビティ	0.6
S	1の集約演算のセレクトビティ	可変
S ₁	2Pの第1フェーズのセレクトビティ	$\min(SN, 1)$
S _g	2Pの第2フェーズのセレクトビティ	$\max(1/N, S)$
G	結果の行数	R S
G ₁	ノード1での結果の行数	R ₁ S ₁
M	ハッシュ・テーブルのサイズ	50M
IO	ページ読出しの時間	0.8ms
t _r	行読出しの時間	200/mips
t _v	行書き込みの時間	200/mips
t _h	ハッシュ計算の時間	200/mips
t _a	集約計算の時間	40/mips
t _m	ページ送信の時間	0.1ms
t _b	ページ・ブロードキャストの時間	(N-1)t _m
mips	CPUのスピード	120/mips
T	ソースデータのスキャン回数	可変
A	1演算当りの集約ファンクションの数	100

なお、最も右側の列の値で、具体的数値が記載されている部分は一例であって、他のコンピュータ・システムでは他の値が用いられる。また、縦の線で囲まれた記号は、その数を示し、縦の線がない記号はそのバイト数を表す。

【0091】幾つかの項目は説明を加えておく。ページ・サイズPは、コンピュータ・システムが取り扱うデータのサイズである。プロジェクトビティpは、リレーションの列のうち用いる割合を示す。1の集約演算のセレクトビティは、グループ数をデータ数で除した値を指す。また、同じ属性に注目する集約演算であっても属性の取扱いが異なる場合、例えば回数をカウントする場合もあれば合計を計算する場合、ある条件を満たす値のみ集約する場合等、もあるので、この異なる取扱いの数をAとする。データ・マイニングのような用途においては、大きな値の場合が多いので表6では100という値にしている。

【0092】以上の前提の下、各アルゴリズムのコストを示す。

【0093】1. 2Pmアルゴリズム

(a) データベースからの読み出しコスト

$$(R_i/P) * IO * T$$

(b) 必要なデータを選択するコスト

$$|R_i| * t_r * T$$

(c) 集約演算するコスト

$$|R_i| * (t_h + t_a * A) * Q$$

(d) 最初のハッシングで処理できなかった分のコスト

$$(R_i * p * Q - M/S_i * T) / P * 2 * IO$$

(e) 結果を格納する行を生成するコスト

$$|G_i| * t_w * Q$$

(f) 送信受信コスト

$$G_i / P * t_m * Q$$

(g) 最終結果を計算するコスト

$$|G_i| * (t_r + t_a * A) * Q$$

(h) 結果を格納する行を生成するコスト (第2フェーズ分)

$$|G_i| * S_g * t_w * Q$$

(i) 最初のハッシングで処理できなかった分のコスト (第2フェーズ分)

$$(G_i * Q - M/S_g * T) / P * 2 * IO$$

(j) 結果をディスクに格納するコスト

$$G_i * S_g / P * IO * Q$$

これらのコストの合計が、2Pmアルゴリズムのコストということになる。

【0094】2. BCmアルゴリズム

(a) データベースからの読み出しコスト

$$(R_i/P) * IO * T$$

(b) ブロードキャストのコスト

$$(R_i/P) * t_b * T$$

(c) 通信バッファから行データを取り出すコスト

$$|R_p| * t_r * T$$

(d) 集約演算コスト

$$|R_i| * (t_h + t_a * A) * Q$$

(e) 最初のハッシングで処理できなかった分のコスト

$$(R_p * p * Q - M / S * T) / P * 2 * I O$$

(f) 結果を格納する行を生成するコスト

$$|R_p| * S * t_w * Q$$

(h) 結果をディスク装置に格納するコスト

$$R_p * S / P * p * I O * Q$$

以上を合計すると、B C m アルゴリズムのコストが計算できる。

【0095】3. R e p m アルゴリズム

(a) データベースからの読み出しコスト

$$(R_i / P) * I O * T$$

(b) 必要なデータを選択するためのコスト

$$|R_i| * t_r * T$$

(c) 宛先を探すためにハッシングし、通信バッファに書き込むためのコスト

$$|R_i| * (t_h + t_w) * Q$$

(d) リパーティションしたものを送信／受信するコスト

$$R_p / P * p * t_m * Q$$

(e) 集約演算コスト

$$|R_p| * (t_r + t_a * A) * Q$$

(f) 最初のハッシングで処理できなかった分のコスト

$$(R_p * p * Q - M / S * T) / P * 2 * I O$$

(g) 結果を格納する行を生成するコスト

$$|R_p| * S * t_w * Q$$

(h) 結果をディスク装置に格納するコスト

$$R_p * S / P * p * I O * Q$$

以上を合計することにより、R e p m アルゴリズムに必要なコストを計算することができる。

【0096】このように各アルゴリズムの実行に必要なコストを計算することができる。これを用いて、以下のような処理(図33)にて実行すべきアルゴリズムを決定する。最初に、ハードウェアに関するパラメータを入力する(ステップ510)。このハードウェアに関するパラメータは、表6のプロセッサ・エレメントの数、ハッシュ・テーブルのサイズ、ページ読み出しの時間、行読み出しの時間、行書き込みの時間、ハッシュ計算の時間、ページ送信の時間、集約演算の時間、ページブロードキャストの時間等、ハードウェア構成から決定するパラメータである。これらのパラメータは、一度コンピュータの構成が決まると変わらないものが多いが、計算によってはプロセッサ・エレメントの数を変更したり、ハッシュ・テーブルのサイズも変更可能である。次に実行する集約演算に関するパラメータを入力する(ステップ520)。表6の項目のうちハードウェアのパラメータとしてのものは、ほぼ集約演算に関するパラメータである。結果の行数等は集約演算のみでは決定しないが、サンプリングを行いグループ数等を見積もることで数値を得ることができる。このような事項は、従来から行われてきたことである(例えば、P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stockes, "Sampling-based esti-

mation of the number of distinct values of an attribute," In Proceedings of the 21st VLDB Conference, pages 311-322, 1995などを参照のこと)。

【0097】このように表6に記されたパラメータが入力されると、先に示したコストモデルにて各アルゴリズムのコストを計算することができる(ステップ530)。その後、その3つのアルゴリズムのうち最小コストのアルゴリズムを決定する(ステップ540)。これにより、決定されたアルゴリズムにて集約演算を実行すれば、3つのアルゴリズムのうち最も高速に演算を実行することができる。

【0098】以上、本発明を実施するために必要とされるコンピュータ・システム及びプログラムに処理フローについて説明した。プログラムの処理フローの各ステップを実行するような回路や、プログラムと回路の組み合わせによって、各ステップを実現する手段を形成することも可能である。各ステップを実現する手段は、各プロセッサ・エレメントの代わりに設け、高速のネットワークで並列処理する。繰り返しを制御するような制御回路を設けること、装置全体を管理するような管理手段を設けることも考えられる。

【0099】また、以上の処理フローを実現するためのプログラムは、フロッピー・ディスク、CD-ROMやMO (Magnet-Optics) ディスク等の記憶媒体上に実現され、流通することが考えられる。さらに、記憶媒体のみならず、有線や無線の通信媒体上で流通する場合もある。このような場合、上述した処理フローの各ステップのうち、幾つかのステップのためプログラムは、記憶媒体、通信媒体にて流通するプログラムとは別途ユーザに提供される場合も考えられる。

【0100】

【実施例】図34は、16ノードのハイパフォーマンス・スイッチを伴うIBM SP2を用いた場合に、どのアルゴリズムを用いるとよいかを示すものである。このシステムの場合、領域aは2 P m アルゴリズムが最適で、領域bはB C m アルゴリズムが最適で、領域cはR e p m アルゴリズムが最適である。演算数と1演算当たりのグループ数が分かれば、最適なアルゴリズムを選択することができる。なお、このような図を作成して最適アルゴリズムを選択する訳ではなく、先に示したコストモデルで各アルゴリズムのコストを計算し、そのコストとの対比にて最適アルゴリズムを選択する。

【0101】

【効果】複数の集約演算を並列に高速実行する方法を提供することができた。

【0102】上記の方法を用いると複数の集約演算を並列に高速実行できるので、OLAPやデータ・マイニングの実行を高速にすることもできる。

【0103】さらに、複数の集約演算を実行する方法を、ハードウェアの条件や複数の集約演算の性質等に基づ

づき、切り換えることができるようにし、様々な条件の下、複数の集約演算を常に同じ方法で実行するよりも、より高速に実行することができた。

【図面の簡単な説明】

【図1】Repアルゴリズムの処理フローを示した図である。

【図2】2Pアルゴリズム、Repアルゴリズム、及びBCアルゴリズムにおける、グループ数対応答時間の関係を示した図である。

【図3】本発明で用いられるコンピュータ・システムの構成の一例である。

【図4】本発明で用いられるコンピュータ・システムの構成の一例である。

【図5】本発明で用いられるコンピュータ・システムの構成の一例である。

【図6】2Pmアルゴリズムの処理フローを示した図である。

【図7】プロセッサ・エレメント1のデータベースの内容を表す図である。

【図8】プロセッサ・エレメント2のデータベースの内容を表す図である。

【図9】プロセッサ・エレメント3のデータベースの内容を表す図である。

【図10】プロセッサ・エレメント4のデータベースの内容を表す図である。

【図11】2Pmアルゴリズムの2番目のステップを図7に対して実行中のプロセッサ・エレメント1のメモリ内の途中結果を示す図である。

【図12】2Pmアルゴリズムの2番目のステップを図7に対して実行したプロセッサ・エレメント1のメモリ内の結果を示す図である。

【図13】2Pmアルゴリズムの2番目のステップを図8に対して実行したプロセッサ・エレメント2のメモリ内の結果を示す図である。

【図14】2Pmアルゴリズムの2番目のステップを図9に対して実行したプロセッサ・エレメント3のメモリ内の結果を示す図である。

【図15】2Pmアルゴリズムの2番目のステップを図10に対して実行したプロセッサ・エレメント4のメモリ内の結果を示す図である。

【図16】2Pmアルゴリズムの集計処理を実行したプロセッサ・エレメント1のメモリ内の結果を示す図である。

【図17】2Pmアルゴリズムの集計処理を実行したプロセッサ・エレメント2のメモリ内の結果を示す図である。

【図18】2Pmアルゴリズムの集計処理を実行したプロセッサ・エレメント3のメモリ内の結果を示す図である。

【図19】2Pmアルゴリズムの集計処理を実行したプ

ロセッサ・エレメント4のメモリ内の結果を示す図である。

【図20】BCmアルゴリズムの処理フローを示した図である。

【図21】BCmアルゴリズムでどのプロセッサ・エレメントがどの集約演算を実行するのかを決定するための処理フローを示す図である。

【図22】図7乃至図10に対し、BCmアルゴリズムを実行したプロセッサ・エレメント2のメモリ内の結果を示す図である。

【図23】BCmアルゴリズムを実行したプロセッサ・エレメント1、3乃至4のメモリ内の結果を示す図である。

【図24】Repmアルゴリズムの処理フローを示した図である。

【図25】Repmアルゴリズムで演算1及び演算2を実行するプロセッサ・エレメント3の処理の途中の状況を説明するための図である。

【図26】Repmアルゴリズムで演算1及び演算2を実行するプロセッサ・エレメント3の処理の途中の状況を説明するための図である。

【図27】Repmアルゴリズムで演算1及び演算2を実行するプロセッサ・エレメント3の処理の途中の状況を説明するための図である。

【図28】Repmアルゴリズムで演算1及び演算2を実行するプロセッサ・エレメント3の処理の途中の状況を説明するための図である。

【図29】Repmアルゴリズムで演算1及び演算2を実行する際に各プロセッサ・エレメントにおいて送信される、演算1関連のデータ及びその送信先を表す図である。

【図30】Repmアルゴリズムで演算1及び演算2を実行する際に各プロセッサ・エレメントにおいて受信される、演算1関連のデータ及びその送信元を表す図である。

【図31】Repmアルゴリズムで演算1及び演算2を実行する際に各プロセッサ・エレメントにおいて送信される、演算2関連のデータ及びその送信先を表す図である。

【図32】Repmアルゴリズムで演算1及び演算2を実行する際に各プロセッサ・エレメントにおいて受信される、演算2関連のデータ及びその送信元を表す図である。

【図33】2Pm、BCm、Repmアルゴリズムのうち、最適なアルゴリズムを選択するための処理フローを示す図である。

【図34】16ノードのIBM SP2で集約演算を実行する際のアルゴリズム切り換え位置を示す図である。

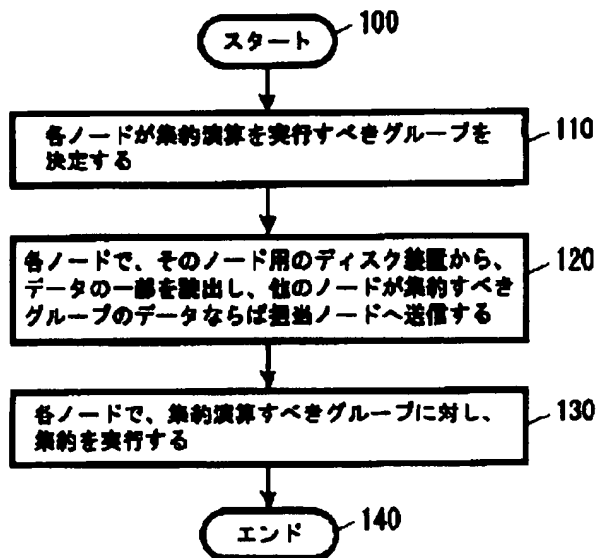
【符号の説明】

1 ネットワーク

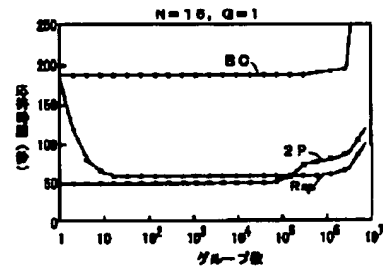
3a, 3b, 3c プロセッサ・エレメント
 5a, 5b, 5c メモリ
 7a, 7b, 7c ディスク装置
 9 メモリ
 9a, 9b メモリ領域

10 メモリ
 11 ディスク装置
 11a, 11b ディスク領域
 13 ディスク装置

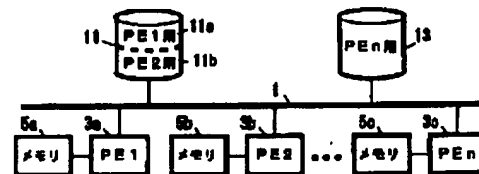
【図1】



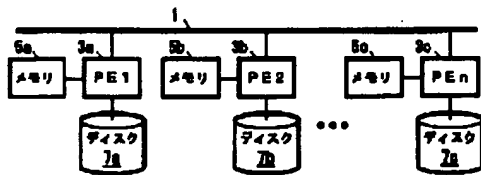
【図2】



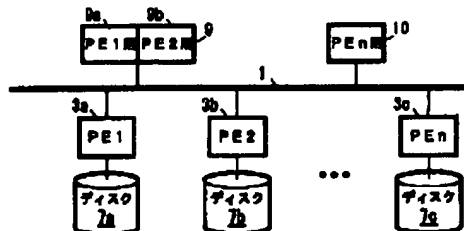
【図5】



【図3】



【図4】



【図7】

ノード1

month	day	week	location	shop#	product#	sell
Apr	7	Sun	Kyushuku	8	2	3
Aug	4	Sun	Hokkaido	13	1	4
Jun	17	Mon	Hokkaido	14	1	2
May	10	Fri	Kanto	13	1	4
Mar	15	Wed	Kanto	6	1	4
Jul	28	Sun	Kyushuku	14	3	1
Apr	23	Tue	Kanto	16	1	9
Mar	15	Sat	Kanto	13	1	8
Mar	15	Sat	Hokkaido	7	2	6

【図8】

ノード2

month	day	week	location	shop#	product#	sell
Jul	8	Wed	Kanto	11	2	6
Mar	21	Thu	Kyushuku	8	3	1
May	6	Mon	Kanto	1	2	2
Aug	2	Fri	Hokkaido	6	1	9
May	3	Thu	Kanto	6	3	7
Apr	23	Mon	Kyushuku	10	2	3
Mar	26	Thu	Kanto	14	3	6
Jul	30	Sat	Kanto	17	3	2
Aug	18	Sun	Kanto	11	1	9

【図22】

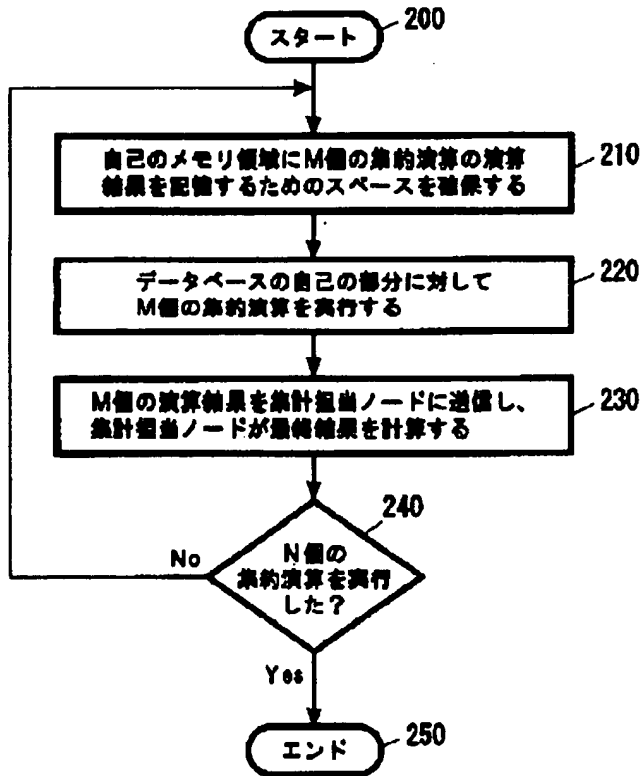
演算2

week	sum(sold)
Mon	7
Tue	26
Wed	26
Thu	26
Fri	27
Sat	33
Sun	25

演算3

location	sum(sold)
Hokkaido	69
Kanto	61
Kanto	32
Kyushuku	28

【図 6】



【図 9】

ノード 3

month	day	week	location	shop#	product#	sold
Aug	3	Sat	Kyushu	8	3	7
Apr	8	Tue	Hokkaido	10	1	8
May	8	Thu	Hokkaido	13	1	8
Mar	1	Fri	Kanto	20	9	8
Apr	30	Tue	Hokkaido	12	1	3
Mar	5	Tue	Hokkaido	1	1	8
Apr	11	Thu	Kansai	5	2	1
Apr	9	Sat	Kanto	5	1	9
Mar	27	Wed	Hokkaido	5	2	1

【図 16】

演算 1

month	sum(sold)
Jan	6
Feb	8

演算 4

product#	sum(sold)
1	188

演算 2

week	sum(sold)
Mon	7
Tue	20

演算 5

location	week	sum(sold)
Hokkaido	Mon	3
Hokkaido	Tue	17
Hokkaido	Wed	10
Hokkaido	Thu	8
Hokkaido	Fri	9
Hokkaido	Sat	8
Hokkaido	Sun	4

演算 3

location	sum(sold)
Hokkaido	85

【図 10】

ノード 4

month	day	week	location	shop#	product#	sold
Jul	10	Wed	Hokkaido	8	1	9
Aug	10	Sat	Kanto	18	3	8
Apr	21	Sun	Kanto	15	3	8
Jan	28	Sat	Kansai	14	2	3
Aug	18	Fri	Kansai	1	3	6
Feb	6	Wed	Kanto	13	1	8
Jan	18	Sat	Kansai	6	3	4
Mar	19	Tue	Hokkaido	13	1	3
Mar	1	Sat	Kyushu	3	1	5

【図 11】

演算 1

month	sum(sold)
Apr	3
Jan	2
Aug	4

演算 2

week	sum(sold)
Mon	1
Sun	7

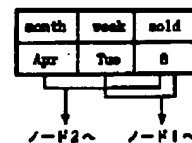
演算 3

location	sum(sold)
Hokkaido	6
Kyushu	5

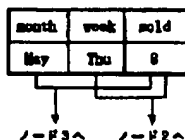
演算 4

product#	sum(sold)
1	6
2	2

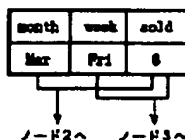
【図 26】



【図 27】



【図 28】



演算 5

location	week	sum(sold)
Hokkaido	Sun	4
Hokkaido	Mon	3
Kyushu	Sun	3

【図 12】

表 1

month	sum(paid)
Mar	10
Apr	13
May	4
Jun	3
Jul	1
Aug	4

表 2

week	sum(paid)
Mon	2
Tue	9
Wed	4
Thu	4
Fri	14
Sat	6

表 3

location	sum(paid)
Hokkaido	13
Kansai	10
Kyushuu	4

表 4

product#	sum(paid)
1	11
2	9
3	1

表 5

location	week	sum(paid)
Hokkaido	Mon	2
Hokkaido	Sat	6
Hokkaido	Sun	4
Kansai	Tue	9
Kansai	Wed	4
Kansai	Fri	6
Kansai	Sat	8
Kyushuu	Sun	4

【図 13】

表 1

month	sum(paid)
Mar	7
Apr	3
May	9
Jun	3
Aug	13

表 2

week	sum(paid)
Mon	5
Wed	6
Thu	14
Fri	9
Sat	3
Sun	9

表 3

location	sum(paid)
Hokkaido	9
Kanto	10
Kansai	13
Kyushuu	4

表 4

product#	sum(paid)
1	10
2	11
3	10

表 5

location	week	sum(paid)
Hokkaido	Thu	9
Kanto	Mon	2
Kanto	Tue	6
Kanto	Sat	3
Kanto	Sun	9
Kansai	Wed	6
Kansai	Thu	7
Kyushuu	Mon	3
Kyushuu	Tue	1

【図 14】

表 1

month	sum(paid)
Mar	10
Apr	11
May	6
Aug	7

表 2

week	sum(paid)
Tue	14
Wed	1
Thu	9
Fri	3
Sat	13

表 3

location	sum(paid)
Hokkaido	10
Kanto	17
Kansai	1
Kyushuu	7

表 4

product#	sum(paid)
1	11
2	3
3	16

表 5

location	week	sum(paid)
Hokkaido	Tue	14
Hokkaido	Wed	1
Hokkaido	Thu	9
Kanto	Fri	3
Kanto	Sat	9
Kansai	Tue	1
Kyushuu	Sat	7

【図 15】

表 1

month	sum(paid)
Jun	4
Feb	3
Mar	9
Apr	3
Jun	3
Jul	9
Aug	15

表 2

week	sum(paid)
Tue	3
Wed	17
Fri	6
Sat	11
Sun	6

表 3

location	sum(paid)
Hokkaido	12
Kanto	15
Kansai	13
Kyushuu	6

表 4

product#	sum(paid)
1	10
2	9
3	11

表 5

location	week	sum(paid)
Hokkaido	Tue	3
Hokkaido	Wed	9
Kanto	Wed	3
Kanto	Sat	3
Kanto	Sun	3
Kansai	Fri	6
Kansai	Sat	7
Kyushuu	Sat	3

【図 17】

表 1	
month	sum(sold)
Mar	45
Apr	44

表 4	
product#	sum(sold)
3	21

表 2	
week	sum(sold)
Wed	30
Thu	30

表 5		
location	week	sum(sold)
Kanto	Mon	3
Kanto	Wed	4
Kanto	Thu	4
Kanto	Fri	6
Kanto	Sat	30
Kanto	Sun	17

表 3	
location	sum(sold)
Kanto	61

【図 18】

図表 1

month	sum(sold)
May	20
Jun	5

図表 2

week	sum(sold)
Fri	27
Sat	23

図表 3

location	sum(sold)
Kansai	50

図表 4

product#	sum(sold)
2	23

図表 5

location	week	sum(sold)
Kansai	Tue	8
Kansai	Wed	10
Kansai	Thu	8
Kansai	Fri	20
Kansai	Sat	15

【図 19】

表 1	
month	sum(sold)
Jul	15
Aug	44

表 4	
product#	sum(sold)

表 2	
week	sum(sold)
Sun	20

表 5		
location	week	sum(sold)
Kyushuu	Mon	3
Kyushuu	Tue	1
Kyushuu	Sat	12
Kyushuu	Sun	4

表 3	
location	sum(sold)
Kyushuu	20

【図 23】

ノ-F1

month	sum(sold)
Jan	4
Feb	8
Mar	46
Apr	44
May	21
Jun	3
Jul	18
Aug	44

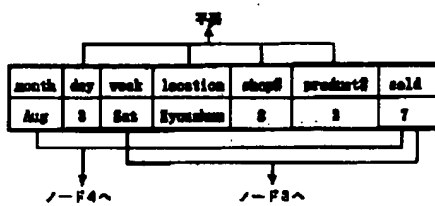
ノ-F4

location	week	sum(sold)
Hokkaido	Mon	2
Hokkaido	Tue	17
Hokkaido	Wed	10
Hokkaido	Thu	8
Hokkaido	Fri	8
Hokkaido	Sat	6
Hokkaido	Sun	4
Kanto	Mon	2
Kanto	Wed	8
Kanto	Thu	8
Kanto	Fri	8
Kanto	Sat	20
Kanto	Sun	17
Kansai	Tue	9
Kansai	Wed	10
Kansai	Thu	8
Kansai	Fri	10
Kansai	Sat	15
Kyushuu	Mon	3
Kyushuu	Tue	1
Kyushuu	Sat	12
Kyushuu	Sun	4

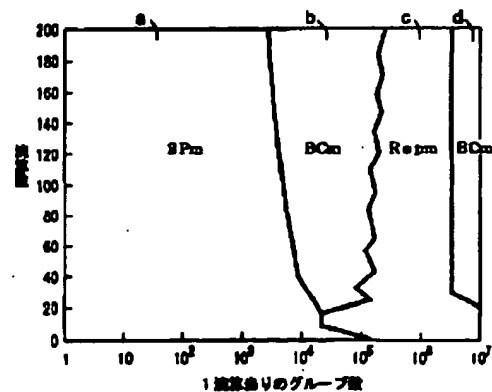
ノ-F3

product#	sum(sold)
1	100
2	21
3	53

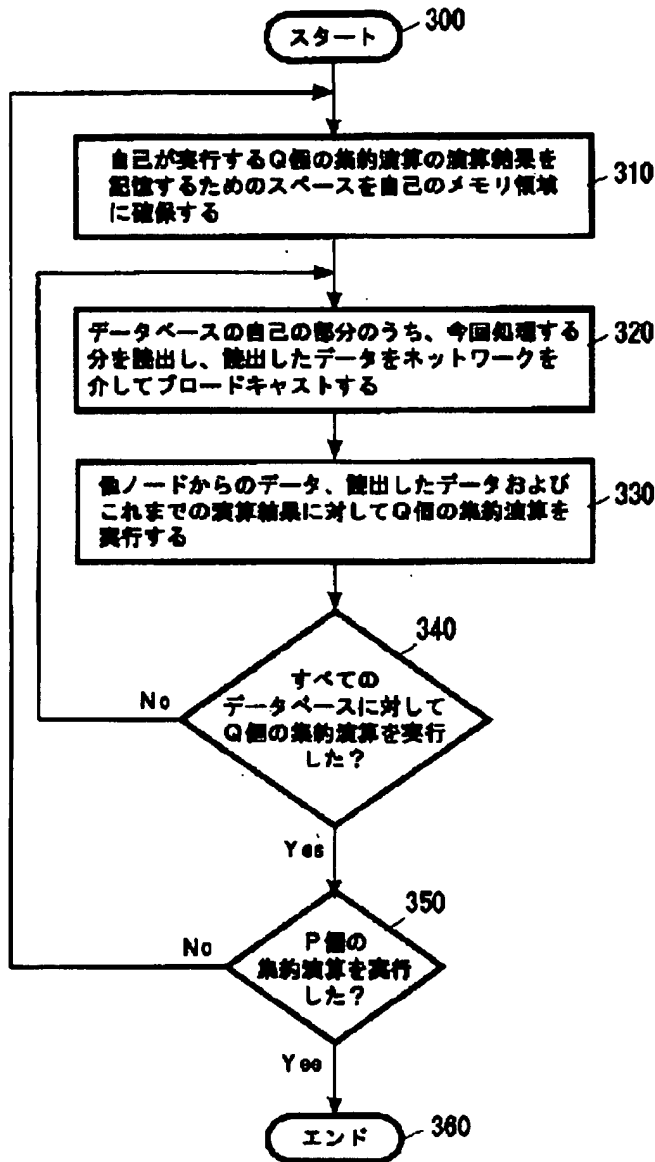
【図 25】



【図 34】



【図20】



【図29】

ノード1			ノード2		
month	sold		month	sold	
Apr	3	⇒ ノード2	Jul	6	⇒ ノード4
Aug	4	⇒ ノード4	Mar	1	⇒ ノード2
Jun	2	⇒ ノード3	May	2	⇒ ノード3
May	4	⇒ ノード2	Aug	9	⇒ ノード4
Mar	4	⇒ ノード2	May	7	⇒ ノード3
Jul	1	⇒ ノード4	Apr	3	⇒ ノード2
Apr	9	⇒ ノード2	Mar	6	⇒ ノード2
Mar	8	⇒ ノード2	Jul	2	⇒ ノード4
Mar	6	⇒ ノード2	Aug	8	⇒ ノード4

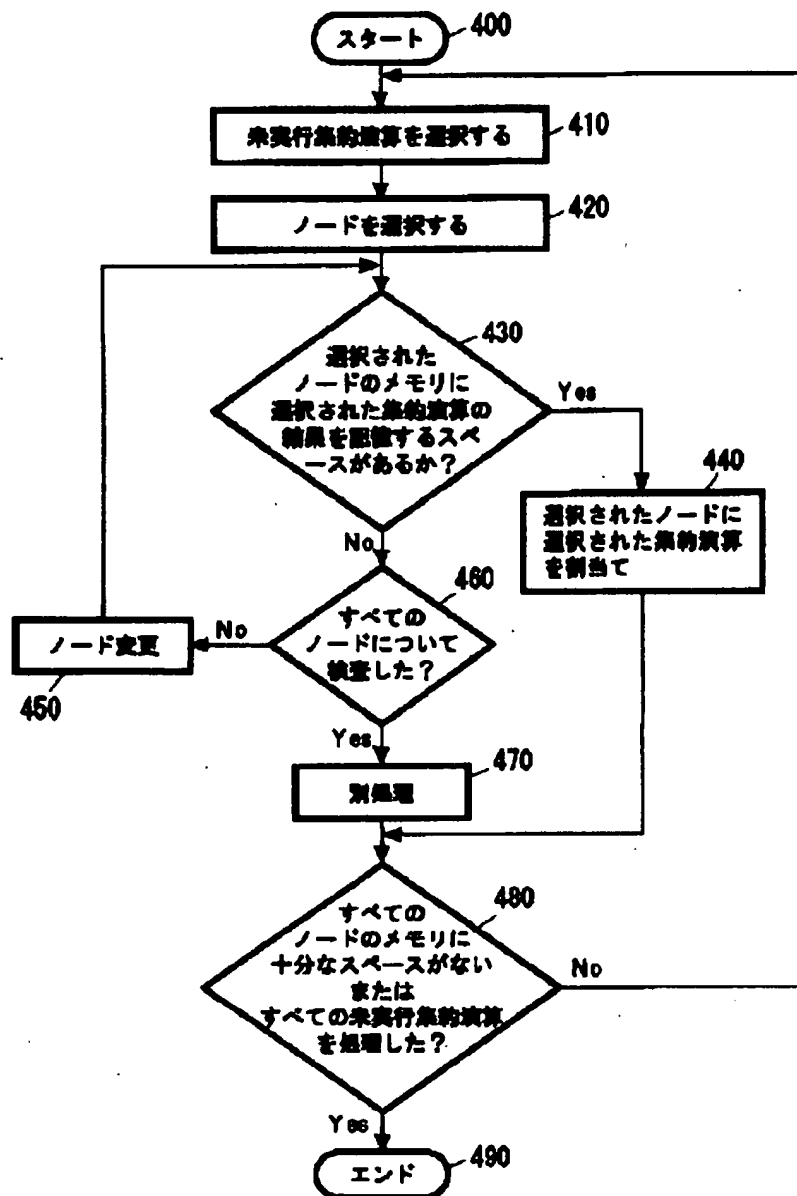
ノード3			ノード4		
month	sold		month	sold	
Aug	7	⇒ ノード4	Jul	9	⇒ ノード4
Apr	8	⇒ ノード2	Aug	9	⇒ ノード4
May	6	⇒ ノード3	Apr	8	⇒ ノード2
Mar	8	⇒ ノード2	Jun	9	⇒ ノード3
Apr	3	⇒ ノード2	Aug	6	⇒ ノード4
Mar	3	⇒ ノード2	Feb	8	⇒ ノード1
Apr	1	⇒ ノード2	Jan	4	⇒ ノード1
Apr	9	⇒ ノード2	Mar	3	⇒ ノード2
Mar	1	⇒ ノード2	Mar	5	⇒ ノード2

【図31】

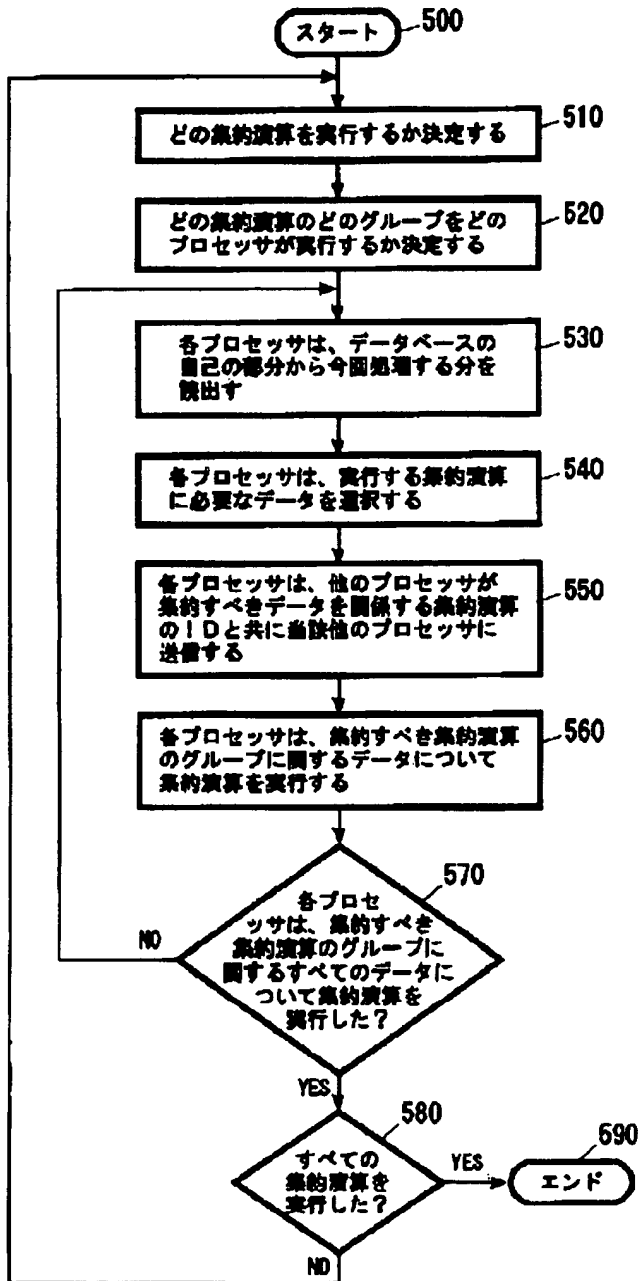
ノード1			ノード2		
month	sold		month	sold	
Sun	3	⇒ ノード4	Wed	6	⇒ ノード2
Sun	4	⇒ ノード4	Thu	1	⇒ ノード2
Mon	2	⇒ ノード1	Mon	2	⇒ ノード1
Fri	4	⇒ ノード3	Fri	9	⇒ ノード3
Wed	4	⇒ ノード2	Thu	7	⇒ ノード2
Sun	1	⇒ ノード4	Mon	3	⇒ ノード1
Tue	9	⇒ ノード1	Thu	6	⇒ ノード2
Sat	8	⇒ ノード3	Sat	2	⇒ ノード3
Sat	6	⇒ ノード3	Sun	9	⇒ ノード4

ノード3			ノード4		
month	sold		month	sold	
Sat	7	⇒ ノード3	Wed	9	⇒ ノード2
Tue	8	⇒ ノード1	Sat	9	⇒ ノード3
Thu	8	⇒ ノード2	Sun	5	⇒ ノード4
Fri	8	⇒ ノード3	Sat	8	⇒ ノード3
Tue	2	⇒ ノード1	Fri	6	⇒ ノード3
Tue	3	⇒ ノード1	Wed	8	⇒ ノード2
Thu	1	⇒ ノード2	Sat	4	⇒ ノード3
Sat	9	⇒ ノード3	Tue	3	⇒ ノード1
Wed	1	⇒ ノード2	Sat	5	⇒ ノード3

【図21】



【図24】



【図30】

ノーフ1		ノーフ2	
month	sold	month	sold
ノーフ4 ⇒ Feb	8	ノーフ1 ⇒ Apr	8
ノーフ4 ⇒ Jan	4	ノーフ1 ⇒ Mar	4
		ノーフ1 ⇒ Apr	9
		ノーフ1 ⇒ Mar	8
		ノーフ1 ⇒ Mar	6
		ノーフ2 ⇒ Mar	1
		ノーフ2 ⇒ Apr	3
		ノーフ2 ⇒ Mar	6
		ノーフ3 ⇒ Apr	8
		ノーフ3 ⇒ Mar	8
		ノーフ3 ⇒ Apr	3
		ノーフ3 ⇒ Mar	3
		ノーフ3 ⇒ Apr	1
		ノーフ3 ⇒ Apr	9
		ノーフ3 ⇒ Mar	1
		ノーフ4 ⇒ Apr	8
		ノーフ4 ⇒ Mar	3
		ノーフ4 ⇒ Mar	5

ノーフ3		ノーフ4	
month	sold	month	sold
ノーフ1 ⇒ Jun	2	ノーフ1 ⇒ Aug	4
ノーフ1 ⇒ May	4	ノーフ1 ⇒ Jul	1
ノーフ2 ⇒ May	3	ノーフ2 ⇒ Jul	6
ノーフ2 ⇒ May	7	ノーフ2 ⇒ Aug	9
ノーフ3 ⇒ May	8	ノーフ2 ⇒ Jul	2
ノーフ4 ⇒ Jun	3	ノーフ2 ⇒ Aug	9
		ノーフ3 ⇒ Aug	7
		ノーフ4 ⇒ Jul	9
		ノーフ4 ⇒ Aug	9
		ノーフ4 ⇒ Aug	6

【図32】

ノーフ1		ノーフ2	
month	sold	month	sold
ノーフ1 ⇒ Mon	2	ノーフ1 ⇒ Wed	4
ノーフ1 ⇒ Tue	9	ノーフ2 ⇒ Wed	6
ノーフ2 ⇒ Mon	2	ノーフ2 ⇒ Thu	1
ノーフ2 ⇒ Mon	3	ノーフ2 ⇒ Thu	7
ノーフ3 ⇒ Tue	6	ノーフ2 ⇒ Thu	6
ノーフ3 ⇒ Tue	3	ノーフ3 ⇒ Thu	8
ノーフ3 ⇒ Tue	3	ノーフ3 ⇒ Thu	1
ノーフ4 ⇒ Tue	3	ノーフ3 ⇒ Wed	1
		ノーフ4 ⇒ Wed	9
		ノーフ4 ⇒ Wed	8

ノーフ3		ノーフ4	
month	sold	month	sold
ノーフ1 ⇒ Fri	4	ノーフ1 ⇒ Sun	8
ノーフ1 ⇒ Sat	8	ノーフ1 ⇒ Sun	4
ノーフ1 ⇒ Sat	6	ノーフ1 ⇒ Sun	1
ノーフ2 ⇒ Fri	9	ノーフ2 ⇒ Sun	9
ノーフ2 ⇒ Sat	2	ノーフ4 ⇒ Sun	8
ノーフ3 ⇒ Sat	7		
ノーフ3 ⇒ Fri	8		
ノーフ3 ⇒ Sat	9		
ノーフ4 ⇒ Sat	9		
ノーフ4 ⇒ Sat	8		
ノーフ4 ⇒ Fri	6		
ノーフ4 ⇒ Sat	4		
ノーフ4 ⇒ Sat	5		

【図33】

